Identification of Function Points in Software Specifications Using Natural Language Processing

Munshi Asadullah

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE 427 :
INFORMATIQUE PARIS SUD

Laboratoire : *Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur (LIMSI)*

THÈSE DE DOCTORAT

INFORMATIQUE

par

Munshi ASADULLAH

Identification of Function Points in Software Specifications Using Natural Language Processing

Date de soutenance : 28/09/2015

Composition du jury :

Directeur de thèse :	Anne VILNAT	Professeur (Université Paris Sud)
Co-directeur de thèse :	Patrick PAROUBEK	Ingénieur de recherche (Équipe ILES, LIMSI - CNRS)
Rapporteurs :	Pascalle SÉBILLOT	Professeur (INSA de Rennes)
	Yannick TOUSSAINT	Directeur de recherche (INRIA Nancy Grand-Est et LORIA)
Examinateurs :	François MARMIER	Maître de conférences (Ecole des Mines d'Albi-Carmaux)
	Sophie ROSSET	Directeur de recherche (Équipe TLP, LIMSI - CNRS)

Abstract

Function Point Analysis is a complex task and identifying function points in a early stage specification document is the most complex part of it. Although it has been in practice for quite sometime, very little research exists regarding automating Function Point Identification and to our knowledge, none exist regarding the use of Natural Language Processing (NLP) for Function Point Analysis. In this day and age, when software development is considered to be a primary human activity, the necessity of a fully automated Function Point analysis is not disputed. We are presenting this work with the objective of establishing a platform for automated Function Point Identification based on Natural Language Processing. In particular, we have presented the details of the development of a corpus for research for Function Point Identification in specification documents. We also presented experimental results obtained using the corpus that can be useful to develop new methods to approach the problem in an objective and reproducible manner. This research and its contributions are expected to lay the foundation stone for the elaboration of research on fully automatic Function Point Analysis from Natural Language specification documents.

Resumé de These

L'entrée dans l'ère de l'intelligence ambiante qui s'accompagne de la nécessité croissante de disposer de logiciels complexes et de grande taille a eu pour conséquence le développement du besoin d'estimer la taille des logiciels pour prédire au mieux le coût et l'effort nécessaires à leur réalisation. Avec la compétitivité qui prévaut dans l'industrie du logiciel, le recours à une estimation précise de la taille des logiciels dès les premières phases de développement est devenu pratique courante. Traditionnellement, des mesures spécifiques basées sur le code source étaient employées. Cependant, avec la prise de conscience par l'industrie du logiciel, du besoin croissant de pouvoir estimer la taille des logiciels très tôt dans son développement, cette activité est devenue une préoccupation de premier plan.

Une fois que le code d'un logiciel est écrit, l'estimation de sa taille et son coût fournissent des informations utiles pour des études comparative ou bien le suivi de productivité. Plus l'estimation peut être effectuée tôt dans le cycle de développement du logiciel, plus les bénéfices que l'on peut en tirer sont importants. L'estimation du coût et de l'effort humain nécessaires pour produire un logiciel produit deux informations essentielles à la gestion du projet et qui peuvent même être calculées avant que la première ligne de code soit écrite. De plus, si l'estimation de la taille du logiciel est effectuée périodiquement pendant le cycle de développement, les gestionnaires du projet vont disposer d'informations précieuses pour l'allocation des ressources et la prévention des risques.

Les recherches que nous présentons ici concernent l'estimation de la taille fonctionnelle d'un logiciel, connue sous le nom d' « analyse en points de fonction » (Function Point Analysis). Dans ce type d'analyse, on cherche à déterminer la taille d'un logiciel exprimée en termes des fonctionnalités qu'ils est supposé offrir d'après le point de vue de l'utilisateur.

Actuellement, le frein principal à l'emploi massif de l'analyse en points de fonction réside dans la nécessité d'avoir recours à des experts humains (les « cotateurs ») qui vont analyser les spécifications en suivant un ensemble de règles de comptage standards. Le comptage

des points de fonction (« Function Point Counting ») est un processus qui nécessite une part importante de main–d'œuvre experte ; il est donc nécessairement coûteux, car les cotateurs doivent lire les spécifications dans leur intégralité avant de pouvoir produire une estimation. À cela s'ajoute le fait que les règles de comptage sont ouvertes à l'interprétation et produisent dans beaucoup de cas des décomptes de poinst de fonction dissemblables lorsque l'on répète une évaluation avec différents cotateurs. C'est pourquoi disposer d'une automatisation, même partielle, du processus de comptage pourrait révolutionner les pratiques.

L'automatisation du processus d'identification des points de fonction dans un document de spécification aura pour conséquence une réduction du temps de lecture nécessaires aux cotateurs pour effectuer leur analyse, accélérant le processus et réduisant ainsi d'autant son coût. De plus l'automatisation permettra d'augmenter la répétabilité des mesures, permettant ainsi d'augmenter la cohérence entre les différentes prise de mesures pour un même projet. À notre connaissance, les travaux présentés dans cette thèse sont la première tentative faite pour analyser automatiquement des spécifications dès la première phase du développement, en utilisant une approche générale basée sur des méthodes éprouvées en traitement automatique des langues naturelles (TALN).

État de l'art

Le champ d'exploration des recherches décrites ici se limite à l'identification des points de fonction dans les documents de spécification en se basant uniquement sur leur contenu en langue naturelle et en utilisant des méthodes issues du TALN. Le découpage et l'éventuelle réécriture du problème nous ont amenés à étudier des méthodes issues de domaines voisins, comme la Recherche d'Information (RI) et l'Extraction d'Information (EI).

Cependant, nous pensons que l'IE peut apporter une contribution significative pour l'identification détaillée des points de fonctions, c'est à dire au niveau de la phrase, et à notre avis établir, même simplement théoriquement, la relation sous-jacente avec ce domaine est important pour les recherches futures. C'est pourquoi nous avons consacré un chapitre (le chapitre 2) à l'IE, alors que le TALN ou l'IR on été abodés à différents endroits du document avec des approches bien établies, dont leurs théories et mises en pratique sont expliquées en détails à chacune de ces occasions.

Dans le chapitre 3 nous introduisons la notion de mesure, ses différentes déclinaisons ainsi que ses fondements théoriques. Nous abordons ensuite l'histoire de l'estimation de la

taille des logiciels avec bien sûr une attention particulière pour l'estimation de la taille fonctionnelle et ses différentes méthodes d'application. Le concept de mesure peut-être vu comme un moyen de quantifier ou de qualifier les propriétés d'un objet afin de permettre la comparaison selon un ensemble de règles prédéfinies. Au quotidien, nous comparons les prix des marchandises, ou la taille des vêtements etc. La science se donne le but de trouver comment mesurer les propriétés d'un objet de manière à ce que les résultats produits facilitent la compréhension de l'objet pour en dériver un modèle permettant de le contrôler ou de prédire son comportement. Et si l'on dispose déjà d'un moyen de mesure établi pour une propriété donnée, la science va s'intéresser à l'amélioration de la qualité, de la précision, de la fiabilité, etc. de la mesure.

L'estimation de la taille d'un logiciel va permettre de quantifier sa taille en des termes qui vont permettre une comparaison détaillée entre les logiciels. Dans le chapitre 3, nous classons les méthodes d'estimation en deux grands groupes, d'une part les méthodes basées sur l'analyse du code source et d'autre part les méthodes fonctionnelles. Nous présentons d'abord en détails les méthodes d'estimation basées sur le code source, puis nous explorons les mesures fonctionnelles qui sont au centre de nos investigations. L'analyse en points de fonction a été proposée par Allan Albrecht comme alternative aux méthodes d'estimation basées sur le code source, lorsqu'il travaillait pour IBM en 1979 (Albrecht, 1979). Sa méthode mesure la taille d'un logiciel en quantifiant les fonctionalités significatives pour un utilisateur. Conçue initialement pour mesurer la productivité de l'équipe de développement, la méthode fût par la suite employée pour estimer la taille des logiciels.

Un des aspects les plus productifs de la méthode est la possibilité de l'appliquer à une très large palette d'environnements de développement et à n'importe quel stade du cycle de vie d'un logiciel, depuis l'élaboration des spécifications préliminaires jusqu'au déploiement sur le terrain. Cependant, l'identification des points de fonction à partir de seuls documents de spécification s'effectue manuellement, ce qui nous le rappelons est un processus long et fastidieux, particulièrement pour les projets de grande taille. De plus, les organisation doivent soit disposer de cotateurs parmi leurs membres, soit avoir recours à des cotateurs extérieurs dont les services sont onéreux. L'analyse en points de fonction repose sur la construction d'une représentation subjective du système d'information décrit dans les documents de spécification, à ce titre elle ne captera que les composents essentiels du système. L'analyse commence par l'identification des frontières conceptuelles logiques du système, c'est à dire les limites de l'application qui est l'objet de l'estimation. Une fois la frontière du système identifiée, les cotateurs vont détailler les fonctionalités décrites dans les spécifications, selon

les cinq types d'entités définies pour l'analyse en points de fonction. Le modèle général utilisé par l'analyse en points de fonction dispose de deux types de « fichiers » (les points de fonction correspondent aux blocs logiques de données) et trois types de « transactions » (les points de fonction correspondent aux mouvements de données). L'automatisation de l'analyse en points de fonction sur des spécifications en langue naturelle pose des problèmes particuliers car jusqu'à présent le développement des méthodes d'analyse a surtout eu pour objectif de restreindre la part de subjectivité inhérente aux cotateurs humains, plutôt que d'exploiter les informations objectives présentes dans les descriptions de systèmes d'information. Néanmoins, les recherches sur l'automation de l'analyse en points de fonction (qu'elle soit partielle ou complète) est relativement ancienne. Les premiers travaux se sont surtout intéressés à identifier les possibilités d'automation du processus ou bien à définir les différents niveaux d'automation. Le chapitre 3 présente un revue détaillée de la littérature sur ce sujet. Par le passé, des tentatives ont été faites pour automatiser l'analyse en points de fonction à partir du code source ou bien à partir d'environnements de développement spécifiques (CASE), voire à partir de document de spécification, mais nous n'avons pas trouvé d'approche basée uniquement sur l'analyse du langage naturel.

Nos Travaux

Les deux contributions principales de nos recherches sont d'une part, le développement d'un chaîne de traitement automatique du langage naturel pour normaliser les documents de spécification selon des critères pertinents pour l'analyse en points de fonction et, d'autre part, la définition d'une méthode d'analyse pour les plus hauts niveaux conceptuels de l'analyse en point de fonction, c'est à dire l'identification des éléments textuels de plus grande granularité, comme les page, les paragraphes etc. susceptibles de contenir des points de fonction.

Si l'on prend en considération le fait que le processus d'analyse en points de fonction est entièrement manuel, simplement identifier automatiquement les pages susceptibles de receler des points de fonction va déjà constituer un amélioration importante du processus d'analyse, car non seulement les cotateurs vont gagner du temps lors de la lecture en se focalisant sur les pages pré–identifiées, mais lorsqu'il y a plusieurs cotateurs, ils vont se focaliser sur les mêmes pages, ce qui va contribuer à la cohérence des mesures lorsqu'elles sont répétées, en diminuant la variabilité sur les donnée d'entrée du processus.

Le corpus du projet, PEC ProjEstimate Corpus) par la suite, a été créé au moyen de la chaîne de traitement linguistique, qui comprend successivement des fonctions : d'extraction de texte, de normalisation, d'analyse syntaxique et d'agrégation des informations produites.

Le projet ProjEstimate

Cette thèse a été rédigée dans le cadre des recherches effectuées par le projet ProjEstimate pour automatiser le processus d'estimation de la taille d'un logiciel. Les travaux de recherche sont financés par ProjEstimate 1 (FUI 13 et labellisation du pôle de compétitivité SYSTEMATIC). C'est un projet collaboratif comprenant des industriels et des académiques dont les objectifs détaillés sont les suivants:

- contrôler le processus d'estimation tout au long du cycle de vie d'un projet.
- fournir les moyens d'appliquer de multiples méthodes d'estimation, y compris les méthodes propres à une organisation.
- réduire de manière significative le coût d'estimation en particulier pour la mesure en points de fonction.
- offrir un infrastructure.
- proposer des fonctionalités d'archivage des différentes estimations associées à un projet.
- pouvoir exporter les résultats d'une estimation vers des outils de génération automatique de rapport.
- disposer d'outils de suivi des utilisateur d'une archive de projet afin de mesurer la capacité d'une organisation à mettres à jour les paramètres d'estimation pour améliorer l'efficacité et la qualité des estimations.

En sus de ces objectifs principaux, le projet ProjEstimate vise à :

- développer une infrastructure de support pour une communauté construite autour de la problématique de l'estimation logicielle et ayant pour but le partage de connaissances et des retours d'expérience ainsi que le développement des collaborations.
- mettre en commun des données anonymes pour servir de référentiel et étalonner les mesures.
- améliorer en continu les logiciels en testant des idées nouvelles visant à simplifier le processus d'estimation, améliorer l'exactitude des estimations et réduire les coûts d'estimation.

Les données qui ont servi à la construction du PEC ont été fournies par deux partenaires industriels du projet, la Banque de France (BdF) et la société PSA Peugeot Citroën Group (PSA). Les données brut sont des documents de spécifications complètes ou partielles, de différent types, par exemple des compilation de besoins utilisateurs, des spécifications

fonctionnelles, des descriptions de traitement hors-ligne (batch), des schéma directeurs de bases de données, des spécifications basées sur le langage UML (Unified Modeling Language) etc. Les formats électroniques sous lesquels se présentent ces données regroupent les formats: Microsoft Word (à la fois l'ancien « .doc » et le nouveau « .docx »), Rich Text Format (.rtf), Microsoft Excel (.xls) et Microsoft PowerPoint (.ppt).

La chaîne de traitement linguistique

Elle est constituée de quatre modules principaux respectivement dédiés à ; l'extraction des contenu indépendamment de la langue (Apache PDFBox), la normalisation du texte, l'analyse syntaxique statistique en dépendances (BONSAI) et l'alignement et la fusion des annotation. Lors de cette dernière étape, les annotations linguistiques produites par l'analyseur syntaxique (les classes morpho-syntaxiques des mots et les paires de dépendances lexicalement ancrées) sont projetées sur les contenus d'origine. Les fonctionnalités de normalisation des contenus et d'alignement des annotations ont été développés spécialement pour ces travaux de recherche. Dans le chapitre 4, nous présenton en détails notre boîte à outils linguistiques (sa conception, les différents processus, les flux d'information et de contrôle etc.) ainsi que les résultat de notre à analyse du PEC. Nous pensons que cette boîte à outils que nous avons conçue pour les besoin du projet ProJestimate constitue une contribution très utile comme base pour de future travaux de recheche sur l'analyse en points de fonction.

L'extraction de texte à partir de contenus électroniques (par exemple extraction du texte des pages web) est un domaine de recherche très actif, mais ici notre objectif était l'extraction d'un quantité de texte et d'information de format suffisante pour nos besoins d'analyse, sans nécessairement chercher tous les raffinement technologiques qui nous auraient sans doute permis d'extraire plus de texte ou du texte de meilleur qualité. Cela cependant au prix d'efforts supplémentaires sans pour autant que le gain ainsi obtenu soit nécessairement significatif, c'est pouquoi par exemple nous avons ignoré les graphiques et les tables contenus dans les spécifications. PDF est un format de document électroniques très répandu d'emploi, sans doute parcqu'il préserve l'apparence originale du document, en effet il conserve dans sa représentation des objets textuels de très bas niveau comme les groupes de caractères, les lignes, les courbes, les images ainsi que les attributs de style de ces différents types d'objets comme la fonte, la couleur, le trait, la forme, la texture etc. Le texte produit par l'extraction avec Apache PDFBox est très bruité puisque tous les éléments textuels sont extraits. Un pré-traitement et une étape de normalisation sont donc nécessaires avant de pouvoir utiliser

le texte extrait. La bibliothèque Apache PDFBox est une boîte à outils codée en Java pour traiter des documents encodés en PDF. Elle est distribuée gratuitement et librement sous la licence Apache License v2.0. Le texte final au format XML est obtenu après une ultime étape de vérification et de formatage avec le parser XML « xmllint ».

Comme nous l'avons déjà remarqué précédemment, toutes les informations de format du texte extraites par PDBBox sont des attributs de forme du texte alors que nous avons plutôt besoin des attributs de structure logique (paragraphes) ou bien linguistiques (mots, phrases). Nous présentons maintenant les éléments principaux de la chaîne de traitement qui va analyser le contenu extraits. Le texte et ses annotations est d'abord transformé en une structure à base de liste au moyen d'un parser SAX disponible en Python (le module xml.sax). Vient ensuite une phase d'élimination des éléments redondants, en particulier les entêtes et les pieds de page. Comme critère de redondance entre deux extraits de texte, nous avons utilise un coefficient de Dice (Sørensen, 1948) seuillé. Si on le considère comme une mesure de similiratié textuelle, pour deux chaînes de caractères x et y, le coefficent peut être défini selon la formule ci-après lorsqu'on l'applique aux bigrammes de caractères,

$$s = \frac{2n_t}{n_x + n_y}$$

La structure à base de liste est ensuite explorée par un module dont la fonction est d'identifier les élément des structures énumératives présentes dans le contenu, puis d'un autre module en charge chargé d'identifier les marques typographiques associées aux éléments de listes (puces, numéros d'item etc.) puisque ceux-ci ne contiennent aucune information utile pour l'identification des points de fonction.

La normalisation du contenu se fait aussi en deux étapes. Lors de la première va produire la segmentation en « tokens » et supprimer les marques typographiques identifiant les éléments de liste. Les tokens sont définis par les deux règles suivantes :

1. Toute séquence de caractères alphanumériques constitue un token individuel.
2. Tous les autres caractères visibles constituent un token par eux-mêmes.

La seconde étape concerne la détection des blocs de texte « improbables », c'est à dire l'identification des blocs de textes qui ont une probabilité très faible, voire nulle de contenir des points de fonction. Ils sont identifiés par le biais de leur distribution des classes de caractères et de quelques motifs filtrants simples.

La construction de la couche initiale d'annotations linguistiques est effectué par le module suivant de la chaîne de traitement qui va assigner à chaque token ses attributs linguistiques.

Ce sont le lemme, les parties du discours détaillée (avec leurs sous-catégories et attributs morphologiques, comme le genre et le nombre pour les noms, ou le temps, le mode et la personne pour les verbes). Nous utilisons aussi un analyseur syntaxique en dépendances (Dependency Grammar) pour déterminer les relations bi-lexicales entre tokens et les frontières de phrases. Le contenu original du texte et la segmentation en token n'étant pas toujours préservés par les outils d'analyse linguistique, loin s'en faut, nous avons implémenté un module de réalignement pour projeter les annotations linguistique sur la segmentation en tokens initiale.

Les données expérimentales

Toutes les expériences décrites dans cette thèse ont été effectuées sur un corpus de spécifications logicielles (18 projets au total) qui ont été fournies par les partenaires industriels du projet ProjEstimate, c'est-à-dire la société PSA et la Banque de France. Dans le cadre du projet, un document de spécification a été annoté en points de fonction par quatre annotateurs membres du projet pour servir de référence.

Les 18 projets du corpus de spécifications ont été analysés par la chaîne de traitement décrite précédemment. Lorsque l'on compare le nombre de phrases et le nombre de tokens, on constate un déséquilibre important entre les données expérimentale d'un part avec un total de 41.937 énoncés et les données de référence avec seulement 901 phrases. Cependant le nombre de lemmes uniques est l'aspect qui nous intéresse le plus. En effet, notre hypothèse pose que les lemmes et les relations syntaxiques les reliant constituent les traits principaux qui vont nous permettre d'identifier les points de fonction. De plus, en comparant la liste des lemmes du corpus expérimental avec celle du corpus de référence, on remarque que sur les 1.438 lemmes de ce dernier, il y en a 1.057 qui sont partagés avec le corpus expérimental.

Les annotations de référence ont nécessité l'établissement d'un ensemble de consignes d'annotation établies au début du projet ProjEstimate en collaboration avec les cotateurs du projet. Les annotations identifies des séquences de mots de quatre types:

1. les « Groupes Fonctionnels » (Functional Groups), qui regroupent plusieurs éléments fonctionnels afin de faciliter le comptage,
2. les « Applications Externes » (External Applications),
3. les « Fonction Transactions » (Transaction Functions)
4. et les « Fonction Données » (Data Functions).

La consigne était que le cotateur devait marquer la plus longue séquence possible de mots indiquant un occurrence d'un point de fonction (seulement un occurrence est annotée lorsque le même point de fonction apparaît dans plusieurs endroits du document, en général la première). Par rapport au début du projet, la procédure d'annotation a été améliorée récemment par l'adpption du logiciel libre WebAnnotator (Tannier, 2012) en remplacement de la fonctionalité de surlignage (balises HTML) d'un simple éditeur de pages WEB.

Parmi les annotateurs du corpus de référence, un est originaire d'ACAPI (cabinet d'experts en points de fonction partenaire du projet) , deux de la Banque de France et un de PSA. Avec les précautions d'usage concernant des statistiques établies sur très peu de données, si l'on constate des similarités de pratique de codage parmi les cotateurs d'une même organisation, on ne peut néanmoins n'être que frappé par la grande divergence dqns la sélections des séquences de mots par les différents cotateurs pour un même point de fonction (parfois même lorsqu'ils sont de la même organisation). Nous avons utilisé comme mesure de l'accord inter-annotaeur, le coefficient Kappa () de Cohen (Cohen, 1960). Le Kappa est une mesure statistique souvent utilisée pour mesurer l'accord entre annotateurs, chacun classant N items parmi C catégories mutuellements exclusives. Dans notre cas le modèle est légèrement différent, puis qu'une entité peut recevoir plusieurs étiquettes de classes, c'est pourquoi nous avons effectué traduction du problème se ramenant à un cas de classification binaire (annoté / non-annoté) pour le calcul du Kappa. Bien que les cotateurs aient annotés les mêmes pages, et étaient d'accord pour au moins un type d'annotation, ils n'étaient pas d'accord entre eux pour tous les types d'annotation.

Sélection des traits d'apprentissage

Pour le premier groupe d'expériences, nous n'avons utilisé que des traits purement lexicaux, c'est à dire avec pas ou peu d'information linguistique. Parmi les traits lexicaux couramment utilisés en Traitement Automatique des Langues, nous trouvons les tokens, les lemmes ou encore les n-grammes de mots ou de lemmes en conjonction avec des caractéristiques textuelles plus générales comme la longue moyenne d'un token ou d'un lemme ou encore leurs nombres d'occurences, ou leurs fréquences relatives. Ensuite, ont été ajoutées les étiquettes morpho-syntaxiques (Parties du Discours, ou Part of Speech enanglais) à différents niveaux de granularité d'information, catégorie morpho-syntaxique, sous-catégorie et traits spécifiques et l'appartenance à une ressource linguistique particulière, par exemple les listes de verbes identifiés par les cotateurs du projet comme potentiellement indicateurs

de points de fonction ou bien des lexiques du domaine public. Et pour finir, nous avons exploré l'apport des dépendances syntaxiques sous la forme de traits construits sur les dépendances syntaxiques, à savoir les traits bi-lexicaux constitués par les d-grammes (Pak et Paroubek 2011), c'est à dire des tuples faits de deux mots reliés par une dépendance syntaxique, potentielement sous-spécifiés pour l'un de ses constituants (mot ou type de dépendance). Les d-grammes s'apparentent aux skip–grams (Guthrie et al., 2006) mais portent plus d'information linguistique. Nous avons aussi expérimenté les «sous–arbres syntaxiques », une extension des d-grammes construite à partir des sous-arbes extraits de l'arbre de dépendances syntaxiques, mais pouvant couvrir plusieurs niveaux de profondeurs de l'arbre et donc plus de deux mots.

Pour étudier les corrélations qui existent entre les distributions de ces traits et les occurrences des points de fonction dans les pages, nous avons générés tous les d-grammes et sous-arbres syntaxiques possibles en utilisant soit l'un soit les deux attributs : lemme et Partie du Discours. Il faut noter que même lorsque nous n'utilisons que des sous-arbres de profondeur minimale et en ne tenant pas compte du type des dépendances, les motifs ainsi construits présentent des avantages certains par rapports aux modèles bigrammes puisqu'eux peuvent capturer des dépendances longue distance.

Les traits lexicaux

Les expériences sur les traits lexicaux ont été conçues pour tester l'apport des ressources lexicales, pour la localisation des points de fonction, par exemple les listes de verbes fournies par les cotateurs. Nous avons choisi deux ensembles de traits qui ont été utilisés dans toutes les expériences à savoir le lemme et la catégorie principale de la partie du discours des mots. Au cours de ces expériences, nous avons aussi étudié l'apport d'informations complémentaires comme la distribution des fréquences d'occurence des éléments (forme, lemme, etc.) des listes d'experts à la fois dans les listes fournies par les différents experts mais aussi dans le corpus, ainsi que le nombre de listes dans lesquelles un élément appraissait. La fréquence d'occurence de chaque lexeme de liste d'expert a été utilisée pour produire par combinaison linéaire pondérée une valeur associée à chaque page, quantifiant la probabilité d'occurence d'un point de fonction dans celle-ci. Dans sa forme la plus simle, la valeur calculée pour chaque page est un indicateur booléen, dont la valeur est vrai si l'un des termes au moins d'une liste d'expert est présent dans la page. L'utilisation d'un combinaison linéaire pondérée a permis d'affiner le classement obtenu en utilisant les fréquence d'occurence des

termes et de définir un seuil minimal pour valider la présence de points de fonction dans la page. Cependant avec cette seconde expérience, les performances se sont dégradée par rapport à la classification booléenne, ce qui peut être expliqué par le fait que la plupart des pages contiennent peu d'occurences et certaines pages ne contiennent que des Fonctions Données plutôt que des fonctions transaction. Pour la reconnaissance de motifs à base de combinaison linéaire, la fréquence d'occurence de chaque lexeme dans les différentes listes d'experts a été utilisée comme poids pour le lexeme et la valeur finale a été filtrée par seuillage a priori. Dans un autre groupe d'expériences nous avons utilisé comme poids les fréquences d'occurence des lexemes dans les données de référence. Dans ce cas, les poids sont le rapport entre la fréquence d'un élément et la fréquence maximale pour les termes de la liste, puis comme précédemment les valeurs obtenues par combinaison linéaires ont été seuillées a priori.

Reconnaissance de motifs basée sur les sous-arbres de dépendance appliquée aux recettes de cuisine

Pour pallier à la petite taille de nos corpus dans le domaine des spécification logicielles dues au contraintes de confidentialité inhérente aux projets industriels, nous avons décidé d'expérimenter notre approche utilisant la reconnaissance de motifs basée sur des sous-arbres de dépendance à des donnée comparable, pour lesquelles des corpus existent. Nous avons ainsi choisi d'utiliser un corpus de recette de cuisines qui a été utilisé pour une tâche d'évaluation ouverte de détection d'ingrédients. En effet une recette de cuisine est d'une certaine manière la spécification des opérations à effectuer pour passer des ingrédients intiaux au plat final. Elle décrit donc un ensemble de transformation des produits initiaux selon un nombre d'étapes précis exactement comme un document de spécification va préciser les différents fonctions de transformation d'information dont l'utilisateur souhaite disposer.

Notre expérience avec les motifs à base de sous-arbres de dépendance décrite précédemment appliquée aux recettes de cuisine s'est révélée être très positive. La méthode que nous proposons utilise des dépendances bi-lexicales orientées extraites du texte des recettes pour en extraire des patrons morpho-syntaxiques. Cest motifs sont ensuite utilisés comme traits dans différentes approches d'apprentissage automatique pour produire la liste finale d'ingédients. Il faut noter que cette approche peut facilement être adaptée à d'autres tâches comparables puisqu'elle consiste à extraire automatiquement des documents des patrons morpho–syntaxiques généraux. À l'exception de l'emploi d'un analyseur syntaxique pour la

langue du document, la méthode est entièrement indépendante du langage. La performance de notre méthode appliquée sur le corpus DEFT2013 (Grouin et al., 2013) a montré de très bon résultats, puisque nous avons dépassé les meilleures performances observées lors de la campagne d'évaluation Nous avons obtenu 0.75 avec la mesure « Mean Average Précision », alors que la meilleure performance observée lors de la campagne a été de 0.66. Dans toutes les expériences que nous avons réalisées avec des traits associé au document entier ont montré de meilleures performances que les meilleures performances observées lors de la campagne DEFT2013.

Apprentissage non–supervisé

L'Apprentissage non supervisé, en particulier pour ce qui concerne le regroupement auto-corellé est une approche d'emploi courant, surtout pour les problèmes de classification supervisées avec des jeux de données d'apprentissage de petite taille mais pour lesquels on dispose de grandes quantités de données non-annotées. L'objectif dans ce cas est d'identifier la structure cachée présente dans les données qui affiche une forte corrélation avec la solution cherchée, sans utiliser aucune connaissance a priori sur la solution ni aucune fonction de valuation ou mécanisme de récompense/punition pour l'apprentissage. Le problème de l'apprentissage non supervisé est très proche du problème de l'estimation de densité de probabilité en statistiques (Jordan and Bishop, 2014), dans lequel on essaie de construire à partir des données observées, une estimation de la fonction de densité de probabilité pour les données cachées. Pour nos expériences, nous avons utilisé abondamment la méthode des centroïdes (« K-Means »).

Tout d'abord, nous avons effectué une analyse en composantes principales (ACP) à la fois sur les donnés brut et les données de référence, en considérent chaque page d'une spécification comme un document autonome. Nous avons utilisé à la fois des traits lexicaux et syntaxiques pour cette analyse. Parmi ces traits associés aux formes de surface, nous trouvons les d–grammes et les dgrammes sous spécifiés qui ont donnés les meilleurs résultats pour faire apparaître les structures cachées. Les motifs d'aggrégation obtenus avec les lemmes et la catégorie principale des parties du discours sont semblables à ceux obtenus avec l'analyse syntaxique qui a cependant fourni les meilleurs résultats. Avec la meilleur compréhension du problème que nous ont permis d ´établir ces expériences, nous avons effectué une réduction des traits d'apprentissage. Nous avons trouvé que l'utilisation des composantes de niveaux supérieurs, voire de tous les niveaux, donne de mauvais résultats, à la fois sur les données de test et les données de référence pour tous les ensembles de traits.

Nous avons donc reparamétré nos expériences et réduit le nombre de traits aux composents principaux identifiés par l'ACP à un très petit nombre de dimensions, de une à quatre, et dans le même temps augmenté la taille possible pour un groupement de 7 à 11. Les données de référence nous ont permis d'etiqueter a posteriori les groupes produits par auto-corrélation, selon les deux classes positive/négative pour la présence de points de fonction. Avec notre environnement d'évaluation, nous avons trouvé que les formes de surface, avec les sous-arbes de dépendance syntaxique et uniquement la première composante identifiée par l'ACP avec un nombre intial de 11 groupements permet d'obtenir les meilleurs résultats. Cependant, ces résultats sont à considérer avec précaution, eu égard à la taille limité des corpus que nous avons utilisé ; ils nous ont néanmoins permis de poser une première hypothèse et d'acquérir une meilleure compréhension du problème de la localisation des points de fonction au niveau des pages des documents de spécification.

Conclusion

Ces recherches nous ont permis d'établir les premiers résultats pour l'identification des points de fonctions à partir des document de spécification en langue naturelle en utilisant une approche uniquement basée sur l'analyse du language naturel. Dans cette première étude, nous avons focalisé notre attention sur l'identification des pages des documents de spécification ayant une plus forte probabilité de contenir des points de fonction, eu égard à la très petite quantité de données disponibles. Ces résultats ouvrent sur un ensemble de nouvelles questions vers l'automatisation complète de l'identification des points de fonction, un problème qui est loin d'être résolu. Nous pensons que nos travaux de recherche fournissent une base utile pour d'autre avancées dans le domaine, en particulier pour ce qui concerne l'apport de l'analyse syntaxique en dépendance et pour la chaîne de traitement des documents de spécification, facilement réutilisable et extensible dans d'autres contextes, et à peu de frais pour d'autres langues.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Today, it is fairly safe to state that almost the whole spectrum of conceivable entities, human made or otherwise, have an associated ***I**nformation **S**ystem (IS)* component, in one way or another. I am not arguing for individual entities (e.g. the rock in my garden) rather the concepts of things. For an extreme example at one end of this spectrum it can be the *New Horizon*[1] mission by *NASA*. The free web service provided by *NASA* allows people from all over the world, to follow the mission progress, enjoy the photos taken during the journey, discuss different aspects of the mission and much more. The extreme aspect is that all this is about a man–made space vehicle (also called *New Horizon*) that is currently (as of 09 July, 2015) approaching *Pluto*[2]. On the other extreme end of this spectrum is virtually everything one can put their eyes on, i.e. day to day communication (e.g. from snail mail to cell phones and communication satellites to fibre–optic back bone), or food (e.g. from automated processing plants to online shopping), or anything else on that matter.

Furthermore, ISs always have a software component associated to them and in the last couple of decades, software and software development have become more structured and evermore complex. Today, software development can be considered to be one of the most significant human activity and thus, the financial component of it gives rise to the necessity of estimating the cost at an early stage of the development for obvious reasons. Cost, however, can only be associated with the unit size of a software. One crucial and absolutely necessary distinction is whether to use the term "*Measuring*" or "*Estimating*" when quantifying the size of a software. Whether it is even possible to measure the size of a software objectively, in our opinion, is a discussion at best philosophical in nature. Of the two concepts, however, *Estimation* is far less

[1] https://www.nasa.gov/mission_pages/newhorizons/main/index.html

[2] $\approx$ 7.5 billion Kilometres from earth and *New Horizon* shall be the closest on 14 July, 2015.

controversial and thus we shall refer to the core objective of this research to be "*Estimating Software Size*". Estimating the size of a software has been somewhat an established practice since the early days of software development, although the emergence of the practice was primarily for making comparison with other pieces of software and evaluating productivity of the development teams. The practice however, becomes an essential component of modern (especially commercial) software development.

Estimating the size of a software and its eventual application in cost estimation is not necessarily a mere financial support tool any more. Overtime it became one of the most important project management tool as well. Often the practice determines the success or failure of contract negotiation and project execution. The deliverables of cost estimation in terms of both financial and management, e.g. effort, schedule, and staff requirements etc., are valuable pieces of information for project formation, management and execution. These factors are used as key inputs for the project bidding and proposal development, estimating budget, staff allocation, project planning, progress monitoring and development control, etc. Inaccurate and unreasonable estimates have been reported to be a major cause of project failure in the *Computerworld*[3] article "Survey: Poor communication causes most ***I**nformation **T**echnology (IT)* project failures"[4]. They reported the *CompTIA*[5] survey of 1,000 IT experts respondents in 2007, finding that two of the three most-cited causes of IT project failure are related with unrealistic resource estimation.

Traditionally, estimation of software was performed from the resultant source code and several metrics were in practice for the task. However, along with the understanding of the importance of code size estimation in the software engineering community, the realization of early stage software size estimation, became a mainstream concern. Once the code has been written, size and cost estimation primarily provides contrastive study and possibly productivity monitoring. On the other hand, if size estimation can be performed at an early development stage (the earlier the better), the benefits are virtually endless. The most important goals of the financial and management aspect of software development namely development cost and effort estimation can be performed even before the first line of code is being conceived. Furthermore, if size estimation can be performed periodically as the design and development progresses, it can provide valuable information to project managers in terms of progress, resource allocation and expectation management. The first such possibility was

[3] http://www.computerworld.com/

[4] http://www.computerworld.com/article/2543770/it-management/survey–poor-communication-causes-most-it-project-failures.html

[5] http://www.comptia.org/

presented by Allan J. Albrecht (Albrecht, 1979), originally for productivity measurement and then as as general sizing metric. This new metric, aptly named ***F**unction **P**oint **A**nalysis (FPA)*, estimates the size of a software as a function of the number of functionalities it is expected to deliver from ***the point of view of the user***. Over the last 35 years the metric has evolved substantially but, core idea and the process remain almost the same as in the original presentation.

One significant problem with FPA is the requirement of human counters, who need to follow a set of standard counting rules, making the process labour and cost intensive (the process is called *Function Point Counting* and the professional, either analysts or counters). Moreover, these rules, in many occasion, are open to interpretation, thus they often produce inconsistent counts. Furthermore, the process is entirely manual and requires ***F**unction **P**oint (FP)* counters to read large specification documents, making it a rather slow process. Some level of automation in the process can make a significant difference in the current counting practice. In this chapter we shall present the problem domain in terms of the task of automation FP identification, our research objectives, important concepts for the proposed solution and the general structure of this thesis.

1.1 The Automation Problem

Traditionally, FPA is performed on software specification documents of different level (e.g. user requirement, functional specification etc.) and at different stages of the ***S**oftware **D**evelopment **L**ife **C**ycle (SDLC)* (e.g. requirement analysis, design, implementation etc.) usually written in natural language by expert users (i.e. having some level of technical competency in software engineering) with different levels of competencies. The availability of different integrated design and development environment (a tool that can be used to design software using numerous design paradigm such as ***U**nified **M**odelling **L**anguage (UML)* and develop the software in parallel e.g. NetBeans[6]), has influenced a significant shift in the overall practice of software engineering and thus the practice of FPA. However, a large number of organizations and software developers still use the traditional methods (i.e. writing specification documents) and use the service of FP counters for FPA. There are three fundamental problems with these scenario, first of all, it takes a lot of time for the counters to go through the documents; secondly, the act of isolating the FPs (regardless of the guidelines being followed), makes is quite difficult to keep the count consistent at

[6] https://netbeans.org/

different stages of the the same project even when using the same document and finally, it is very expensive. Automation of the process of identifying the FPs in a document accurately, will at least reduce the reading requirement of the counters, making the process faster and thus shall significantly reduce the cost. Moreover, consistent identification of FPs will allow the production of consistent raw function point counts. To the best of our knowledge, the work presented in this thesis is an unique attempt to analyse specification documents from early stages of the *SDLC*, using a generic approach adapted from well established ***N**atural **L**anguage **P**rocessing (NLP)* practices.

A survey presented by Mendes et al. (1996) listed almost all the commercially available tools for different stages of the FPA process that were available at the time (52 tools). The authors classified the tools into 10 categories and some tools were assigned in multiple categories. The distribution showed that 8 out of 52 products claim to be able to provide some level of automatic FP counting, but the authors reported that only 4 products (mere 3% for the redundant list) of the tools focus primarily on the problem of reliable automatic FP counting. All of them unsurprisingly, were bound to some specific system platform or used some sort of ***C**omputer **A**ided **S**oftware **E**ngineering (CASE)* tool with specialized design and development paradigm and often performed the count from the source code. Moreover, the 18 steps automation that the literature adopted (Mendes et al., 1996), all the systems were reported to have required some level of human intervention, for at least one of the steps (especially to identify the target system) and none of them were capable of using specification documents written in ***N**atural **L**anguage (NL)*.

In our research we could not find any recent survey, however, there is the ***A**utomated **F**unction **P**oints (AFP)* specification[7] published by the ***O**bject **M**anagement **G**roup (OMG)* in January 2014 that provides clear guidelines for the automation of FP counting following the ***I**nternational **F**unction **P**oint **U**sers **G**roup (IFPUG)* counting practice. The current version (version 1.0) is primarily specified FP counting from source code and database components rather than being applicable to early specification. The commercial attempts to use the OMG specification are already in progress, e.g. The ***C**onsortium for **I**T **S**oftware **Q**uality (CISQ)* is leading The ***A**utomated **E**nhancement **F**unction **P**oint **S**pecification (AEFP)* project[8] since 2015 and ***CAST*** AFP tool[9]. The observation here is that although, there is a clear necessity for automated FP counting from specification texts, and to our knowledge the research on the

[7] http://www.omg.org/spec/AFP/1.0/

[8] http://it-cisq.org/cisq-to-start-work-on-automated-enhancement-function-point-specification/

[9] http://www.castsoftware.com/products/automated-function-points

subject is virtually non-existent. Automation of FP identification poses significant amount of difficulties and the key problems that we have identified ate the followings,

- The most significant problem is the absence of clear and unambiguous counting guidelines that can allow the development of knowledge based automated tool for FP counting.
- In the ever more cautious industrial practices of our time, most specifications are often considered to be classified contents, thus availability of large amount of documents especially from a single organization (in fact we are focusing on large organizations) that can provide coherent documentation and clear pattern is hard to come by.
- Existing counting practices often do not give any significance to the documentation of the counting process itself i.e. counters do not document the source of the FPs as they record the results thus identifying the correlation between text and FP is very difficult.
- Due to organizational culture and the inherent ambiguity that exists in the FP identification process, the confidence in any annotated data is quite low for this specific task. Thus defining an objective and consistent annotation guideline is a concern before establishing an automated system since the evaluation for such a system can only be performed in a subjective manner.
- Specification documents often contain important information such as database design in tabular form or as diagrams. This type of information is quite difficult to process within the NLP paradigm and are often subjects of detail research itself. Moreover, list elements in a document often contains list of actions that are quite important in FP detection (especially in the detection of transaction functions), are again found to be rather difficult to process.
- Most specification documents are not directly machine readable, thus extraction and segmentation often introduce large amount of noise in the original data and text extraction and segmentation have been heavily reviewed in the NLP research community for the seer complexity of these problems.
- Human readers identify FPs by understanding the text, the tables and the graphics together. However, automated systems dealing with these diverse types of information sources are scarce if not absent and the state–of–the–art performance of such systems are anything but satisfactory.
- Text understanding is very important in FP counting, but, automating the process requires multiple levels of preprocessing e.g. tokenizing, parsing, word sense disambiguation etc. These tasks themselves are active fields of research and each of these sub–tasks depends of some other lower level task, thus, every stage of the pre–processing is expected to introduce additional noise in the data.
- Finally, the lack of labelled data also makes it virtually impossible to use well established supervised ***M**achine **L**earning (ML)* methods and perform objective evaluation of any AFP identification system.

An automated system for the task of FP identification is more likely to be put under scrutiny for one other important issue: performance. NLP tasks are often designed to be a process chain, rather than a single process performing all the tasks. This practice significantly reduces the performance since it is rather hard to perform parallel processing of different sub–tasks due to their dependencies on other tasks. This research will attempt to provide an overview of the problems and possible directions towards an automated FP identification from specification documents.

1.2 Research Goals

The difficulties associated with the automation of FP identification are non–trivial because a single research initiative is not enough to address the difficulties associated with each of the intrinsic sub–components of FPA. The best course of option is to recognize a reasonable framework and use state–of–the–art resources for each of the cogs of the proposed solution. We have proposed three hypotheses that, in our opinion, summarize the general features of the proposed framework,

Hypothesis 1: FP identification, from a higher order textual group perspective (i.e. page or paragraph level), is a specialized ***I**nformation **R**etrieval (IR)*, more specifically, *document classification* task.

Hypothesis 2: Individual function identification however, is a specialized ***I**nformation **E**xtraction (IE)* task, more specifically a slot–filling IE task where slot values need to be filled with FP core elements (namely data functions and transaction functions).

Hypothesis 3: Some level of linguistic knowledge engineering is necessary to approach this problem however, deep linguistic analysis may not be necessary, even perhaps counter productive.

The objectives of this research is to find potential methods for identifying FP component in written text within the limitations of the NLP tools available for performing the sub–tasks of FPA. The solution is expected to have the following two prime objectives,

- Minimizing the amount of reading content and thus reading time for the human counters.
- Maximizing the consistency of the FP count through systematic identification.

These two features are loosely interdependent, thus making improvements on one of them is expected to influence the other upto a certain degree. Our research goals thus can be summarized as follows

- Extraction of software specification documents in electronic format (e.g. **.pdf** or **.doc** format documents) into a machine readable format (e.g. *E**x**tensible **M**arkup **L**anguage (XML)*). The extraction process is expected to be accompanied by a post–processing tool to clear some of the noise from the data.
- Applying a basic linguistic analysis process chain on the extracted data to incorporate multiple layers of linguistic knowledge and labels on the original text. NLP techniques will be used at this stage performing tasks such as tokenization, ***P**art-**o**f-**S**peech (POS)* tagging, parsing with dependency relation extraction etc.
- Reducing the number of pages a human counter has to go through to identify all existing FPs in a given document.
- Accommodate the identification of text segments (e.g. words, or sentences or pages) containing FPs in a given text.

We shall demonstrate the possible means of achieving the research goals and test our hypotheses, presented earlier, in the process. Some fundamental concepts that we consider to be an integral part of this research will be addressed in the next section.

1.3 Concepts

Our problem domain can be defined as the analysis of software specification documents and FP identification in an automated manner. Software specifications are special cases of the general specification documents. Since, such documents are expected to contain the *process description* to perform a task and by definition, FPs are counts of the functionalities provided by a software, we can infer that at the top level at least, it is a *process* identification task. We shall primarily investigate the possibility to use basic textual features e.g. token length, token n-grams etc. and very fundamental linguistic features such as lemmas and POS. However for advanced analysis syntactic patterns shall also be introduced. Following Faure and Nédellec (1999), we have assumed that verbs play a fundamental role in *process* detection and extract objects that are connected to them as subjects or direct objects, in particular for verbs that are in imperative mood. From a syntactic and semantic point of view, we are looking for common structures involving a verb and its arguments, among which we are interested in the subject, patient and recipient arguments of a verb. To establish the "*object*" and "*process*" we used the bi–lexical dependency relations using a framework adopted from *d–gram representation* (Pak and Paroubek, 2011). In this section we shall give an overview of these terms.

1.3.1 Specification

> **Specification**
> *noun* | *spec·i·fi·ca·tion*
> `A detailed description of work to be done or materials to be used in a project : an instruction that says exactly how to do or make something.`
> – Merriam–Webster Online dictionary

A large number of domains can crawl under this umbrella definition (e.g. cooking recipes, software specification, description of experiments, instruction manuals etc.). We are not suggesting that specification is a category of sentences, a taxonomical class (specification sentences), coined by Higgins (1973). We are in fact in total agreement with Heycock (2012) in this aspect that a "specification" as a class for sentences is rather unnecessary. However, it is reasonable to assume that the type of text to be found in a specification document uses a finite number of sentence patterns to express the desired action descriptions. We thus hypothesize that a finite variation of morpho-syntactic pattern must exist to express process-object interaction. First, we need to establish the theoretical basis of events, processes and their interaction with objects.

1.3.2 Objects, Events and Processes

In this section we shall not try to provide the philosophical or ontological basis for these concepts, rather present the extent of these concepts that have been used in our research. A detailed study on objects and events and process was presented by Galton and Mizoguchi (2009), and we shall present most of the definition according to this work. However, the granularity of details for these concepts required for our research is much coarse and it shall be reflected in the following discussion. These concepts have already been explained by Asadullah et al. (2014a) to identify ingredients from cooking recipes. There is an uncanny similarity between the tasks since by definition FPs are expressed in terms of *verbs* (since having a functionality either implies having a state defining or action defining verb present almost without exception). Many of the examples thus, are derived from the cooking recipe domain and are expected to represent a concrete picture of these concepts. We shall present the work (Asadullah et al., 2014a) in details later (see § 5.4.2). Furthermore, understanding these concepts and establishing their scope is crucial for this research.

Objects are spatial elements i.e. something that occupies physical space, that can change over time (e.g. from egg to boiled egg), but do not have any temporal part (e.g. an egg

after one hour is still an egg). According to Galton and Mizoguchi (2009) objects can have spatial parts, but those parts are not the same object, rather matters or different objects (e.g. half of an egg is not an egg). However, This relation has little impact in our work (e.g. one may need only egg white for a recipe but the ingredient may still be an egg). Therefore, we restrict our focus to normalized concrete objects and their interaction in a process or an event. Events and processes are difficult concepts to put one's fingers on precisely. Both events and processes has been described in (Galton and Mizoguchi, 2009) as an action over time, where processes describe an action without defined temporal boundary and events, with defined temporal boundaries i.e. start and end. From the cooking recipe a definitive example can be "boiling an egg for 15 minutes". By definition, this is an event i.e. start boiling an egg and finished after 15 minutes. Any intervals in between, by definition are not events, thus are processes. Another property of the events and processes is that they can have spatial components i.e. involvement of objects in the action. In the cooking recipes all the processes and events are thus expected to be informative. For our research these stringent definitions have little importance, thus events and processes have been used as interchangeable through out the whole thesis. However, these definitions can be useful in the adaptation of the method for some other tasks.

1.4 ProjEstimate Project

This thesis is produced as a part of the ongoing research for finding an automated means of analyzing specification text to automatically estimate the size of a software. The research is conducted and funded by the *ProjEstimate Project*[10] (FUI 13) currently running under *SYSTEMATIC*[11], the *Paris Region System and ICT* cluster. The project comprises of an organization body and several partners from both academic and industry domains. *Estimancy*[12] is the current project leader and the partners of *ProjEstimate* are,

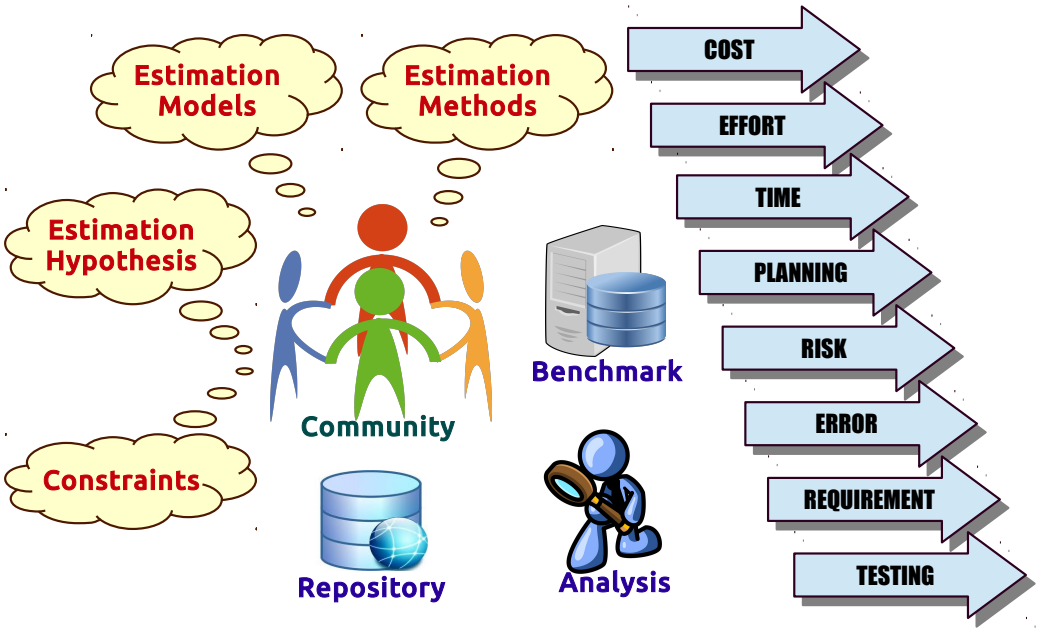
- Two large companies,
  - *Banque de France (BdF)*[13]
  - *PSA Peugeot Citroën*[14]
- Three small and medium enterprises,

[10] http://www.systematic-paris-region.org/en/projets/projestimate
[11] http://www.systematic-paris-region.org/en
[12] http://estimancy.com/
[13] https://www.banque-france.fr/
[14] http://www.psa-peugeot-citroen.com/

  - *Estimancy*
  - *ACAPI*[15]
  - *Sparkom*[16]
- One public research laboratory – *LIMSI*[17]
- One research institute – *ARMINES*[18]

The primary objective of the project is to develop a complete system that will provide it's customers a solution to better control software engineering projects and software systems through reliable software development estimators i.e. size estimator, cost estimator, productivity monitoring etc. The challenge is to improve estimation practices to reduce cost and the implementation of estimation methods and tools in a single platform. The research presented in this thesis is part of the responsibilities of *LIMSI* as a partner of the project. Figure 1.1 presents a general overview of the project.

Fig. 1.1. ProjEstimate Project Overview



The specified goals of the project in terms of the solutions expected to be provided to an organization developing information systems are summarized below,

- Providing control over the estimation process throughout the life–cycle of a project.

[15] http://www.acapi.fr/
[16] http://www.sparkom.com/
[17] http://www.limsi.fr/
[18] http://www.mines-albi.fr/

- The provision to apply multiple estimation method including the specific methods adopted to comply with the practices of an organization.
- Significant reduction in the estimation cost, especially for functional measurement.
- Infrastructure for peer–based support system.
- Repository infrastructure to store various estimates of a project.
- Export estimation results to report generation tools.
- Analysis tools to analyze repository entities to effectively measure the performance of an organization to update estimation parameter to improve efficiency and quality.

In addition to these primary goals, ProjEstimate project has a set of complementary goals,

- Infrastructure development for a community to ensure mutual support and knowledge, experience, ideas, training etc. sharing,
- The provision for anonymous data sharing for benchmarking.
- Continuous improvement of the software by stimulating areas of research to explore the possibilities to simplify estimation process, improve accuracy and reduce costs.

Table 1.1. Work Packages for The *ProjEstimate* Project

WP1	Preliminary selection and data collection for the project – includes preparatory tasks and data collection to accumulate the information necessary to initiate the project
WP2	Data analysis models – bringing together the theoretical research necessary for the data analysis for *ProjEstimate*. This work should provide methods and algorithms for data management services.
WP3	Tool: Specification – includes the tool development for the *ProjEstimate* platform.
WP4	Tool: Framework – includes the tool development for the *ProjEstimate* framework.
WP5	Tool: Module – includes the development of the different module for *ProjEstimate* project.
WP6	Deployment and Testing – includes the deployment and testing of the *ProjEstimate* platform.
WP7a	Community – the creation and maintenance of the *ProjEstimate* community.
WP7b	Services – deployment of the online services for the *ProjEstimate* project.
WP8	Project management – The administrative aspects of the *ProjEstimate* project.

The top level ***W**ork **P**ackages (WP)* distribution among the partners of the project (which also provides the overview of the general work flow of the project) is presented in Table 1.1. Among the WPs, *LIMSI* contributes significantly in WP2, WP5 and WP6. This research is strongly associated with WP2, i.e. data analysis for the project and WP5, i.e. component module development. We have significantly contributed in developing the prototype for the modules needed for requirement document analysis, identifying FPs in text written in *natural*

language and generate data for the task and provide suitable representation of the data. The following section will provide the organizational overview of this document thus, shading some light on how our contribution shall be presented (maintaining the parallel to the goals of the *ProjEstimate* project).

1.5 Organization

The scope of this research is limited to the identification of FPs in the specification documents within the NLP framework. The breakdown and eventual reconstruction of the problem domain led us to explore other methodologies commonly practised in domains such as IR and IE. However, in our opinion, IE is a significant theoretical possibility for detailed FP identification (i.e. at sentence level) and to establish that underlying relationship, theoretically at least, was important for further advancement of the research. Thus a detailed chapter is dedicated to IE, whereas, general NLP or IE was used at specific parts of the research and usually already established methods were used, thus the methodologies and the related theories were addressed locally and with necessary details. The structure of the chapters can be summarized as follows,

- In Chapter 2 we present the state–of–the art of IE research. Although, IE concepts have not been used extensively during the experimentation, in our opinion, the problem domain is best explained within the boundary of *NLP* and both *IE* and *IR* can play an important role in finding a solution. *IE* has been a rather independent research field in terms of having generic methods and well established concepts. However, *IE* tasks have always been problem specific with limited generic methodology and in our opinion, less clearly established concepts. This chapter will try to provide an overview of the history, practice and task definitions for IE and its relation to the problem domain.
- In Chapter 3 we introduce the general concept of measurement, types of measurement etc. i.e. the theoretical basis for the concept. We then present the problem domain by introducing the concepts necessary to understand the domain, i.e. software size estimation, history and the progression of the size estimation theories and practices. Functional size measurement and different practices will also be presented. We have concluded the chapter introducing our contribution in improving the state–of–the–art of the problem domain.
- In Chapter 4 we introduce our primary contribution by introducing the dataset and the tools and the process of corpus generation from the specification documents. We also present a detailed analysis of the corpus in terms of lexical, morphological and syntactic relation (dependency relations) distribution. We also present the available expert annotated data from our experts.

- Using the aforementioned corpus and annotated data, in Chapter 5 we present the experiments that we ran to understand the underlying pattern in the corpus and its correlation to expert annotations. We also demonstrated how different features correspond to existing patterns in the data by using ***P**rincipal **C**omponent **A**nalysis (PCA)* and unsupervised learning algorithms.
- Finally, in Chapter 6 we present a brief summary of our experience with the whole research and our interpretation of the experimental results and possible future directions for perusing this research.

Chapter 2

Classification & Information Extraction

The availability of large amount of digital data during the last couple of decades opens a whole world of opportunities for the computational ***I**nformation **R**etrieval (IR)* and ***I**nformation **E**xtraction (IE)* research. A lot of the information need in practical situations (e.g. online news, business reports, legal documents, medical records etc.) requires automatic processing of text to both identify relevant documents and extract the important information from those documents. As a matter of fact, one can argue that the emergence of IR and IE as independent disciplines is the direct outcome of ever growing information need complimented by mass digitization of printed content and mass production of digital text. ***N**atural **L**anguage **P**rocessing (NLP)* techniques have also been excelled in terms of performance and diversities for somewhat the same reasons. Over the years IR and IE have become disciplined and well structured fields of research. Along with different ***M**achine **L**earning (ML)* methods NLP based methods have been used extensively in these researches.

Our research will explore the utilization of both these discipline but with more emphasis on IR. One of our research objectives is to identify pages that contain ***F**unction **P**oint (FP)* description. Using ML algorithms we shall effectively establish an IR method for text classification. In our case, it would be page classification since pages are the logical blocks of text being our object of interest. We are also interested in expanding our method to smaller logical text blocks such as, paragraphs (or fixed size window if actual paragraph segmentation is not possible) and sentence. However, we recognize that the identification of sentences with FP description will be beyond the scope of IR applications and it can be better handled with IE methods. Although, we did not perform any experiments for sentence identification or explicitly used any IE method, it is the most probable future direction of this research and

thus we briefly present the state–of–the–art of IE research. We also presented the rationale for not adopting IE as the primary focus of the research presented in this thesis.

2.1 Information Retrieval & Text Classification

Information Retrieval encompasses the broad classes of tasks and methods with the objective of identifying text blocks that meet a given search objective. Technically, when one is looking for the phone number of a friend and find it in a phone book, it is a form of IR. The example is very simple and gives the impression of the processes used in any standard database search. However, *IR* research as an independent discipline, historically distance itself from the notion of database search (i.e. query) since databases are considered structured representation of data and retrieval primarily depends on forming the right search query. To give the proper notion of IR for us, we are inclined to adopt the definition given by Manning et al. (2008),

> *Information Retrieval (IR)* is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). – Manning et al. (2008)

IR tasks often deal with sources of different structural levels e.g. unstructured (free flowing text), semi–structured (XML), fully structured (relational database) or even hyper–linked (web content). Furthermore, the source media can be of many different types e.g. text, 2D/3D graphics, audio, video etc. even combination of *two* or more types (Doyle and Becker, 1975). The primary information access methods often utilized can be short–listed: ad-hoc retrieval (i.e. one time query such as web search), filtering (usually by maintaining a constant search profile for applications such as spam filtering), classification (grouping information into predefined classes) and topic detection and tracking (e.g. clustering news in a stream). There are many more aspects of *IR* that can be crucial for different applications e.g. context for application such as ***Q**uestion **A**nswering (Q&A)* or time dependence (when dealing with continuous stream of information source such as news feeds). However, for this research we are interested in ad–hoc binary classification type information access for *text source*. Our objective is to classify a set of sources (e.g. documents or blocks of text) based on the information requirements (i.e. query).

The computational retrieval of information began in the late 1940's (Manning et al., 2008) but the term *Information Retrieval* was coined much later by Mooers (1950). The necessity

arise from the advancement of computer technology[1] and the availability of large amount of digital data[2]. Furthermore, the availability and extensive usage of web search engines on a regular basis puts *IE* in the center of many information intensive researches. Many of the pre–computerized management of large amount of information evolved from the discipline of librarianship (Sanderson and Croft, 2012), usually for items like books or articles and in the form of indexing using cataloguing schemas. However, this approach can be traced back to thousands of years[3] (Eliot and Rose, 2009, pp. 90). There has always been the need of IR and over the course of history many different methods have been proposed. In the following sections we shall explore the chronological progression in *IR* along with a general overview of the IR methods with our interests in focus.

2.1.1 Pre–Computerized IR

The early requirements of *IR* were rather primitive, primarily concerned about indexing and categorizing documents (books, articles etc.) and find them when necessary and cross–referencing items on the basis of two or more categories. One of the earliest example can be the 1920 patent (Edward, 1920), where catalogue card with holes (each related to categories) that can be aligned in front of each other to find if any entry in a collection related to the search categories. Beyond the crude methods, mechanical devices that can search a catalogue for a particular entry had also been devised. One of the earliest example can be the patent filed by *Emanuel Goldberg* back in the early 30's (Goldberg, 1931). The device can search for a pattern of dots or letters across catalogue entries stored on a roll of microfilm. Other devices based on the same principal were developed, such as the searching with microfilm by Davis and Draeger in 1935 described by Mooers (1959) and later the *Memex* system, a film–based searching machine proposed by *Vannevar Bush* in 1945 (Bush, 1945). Shaw (1949) also proposed a device to search through a $2,000$ foot reel of film. Each half of the film's frames had a different purpose: one half for frames of material and the other half for index entries. He reported that the selector was able to search at the rate of $78,000$ entries per minute. Other methods were also explored such as, punch card, lights and photocell based system by Luhn (C&EN, 1954) that could match a consecutive sequence of characters within a larger string. As a matter of fact, the system presented in Mooers (1950) was also a punch card based system and according to Jahoda (1961) the mechanical systems continued to be

[1] "The number of bits of information packed into a square inch of hard drive surface grew from $2,000$ bits in 1956 to 100 billion bits in 2005" (Walter, 2005)

[2] One can see the historic growth of text in http://userwww.sfsu.edu/fielden/hist.htm

[3] *Callimachus*, a 3^{rd} century BC Greek poet to be the first person known to create a library catalogue

developed until the computer based systems surpassed the effectiveness of any mechanical system for *IR*.

2.1.2 Early Computer Based IR

One of the earliest example of computer based IR was presented by Holmstrom (1948) describing a machine called the *UnivAC* that is capable of searching for text references associated with a subject code. The code and text were stored on a magnetic steel tape and it was reported to be able to process 120 words per minute. Mitchell (1953) also described project using *UnivAC* to search one million records indexed by upto six subject codes and the estimated time for a search was reported to be about 15 hours. One can find a detail study of the *IR* systems developed during the 1950's in an article by Nanus (1960). Another review by Brownson (1960) also summarized IR implementations in the 1950's including the work done in the former *Soviet Union*. The two significant development at this stage allowed *IR* became the specialized discipline it is today: *indexing* the documents in a collection and systematically *retrieving* them.

The traditional approach of *indexing* was to group documents based on the topic(s) assigned to it using hierarchical subject classification scheme, such as the *Dewey Decimal* system (Dewey, 1876). Alternative to this indexing was also proposed, for example, the *Uniterm* system (Taube et al., 1952) proposed to index items by a list of keywords. Cleverdon (1959) conducted a detailed comparison of retrieval effectiveness using *Uniterms* and the more classic indexing techniques. After extensive scrutiny (Cleverdon, 1991) the results of the original study was found to be accurate and many aspects of the "*test collection"* approach to evaluation proposed by Cleverdon are still in use for both academic research and commercial search testing today (Sanderson and Croft, 2012).

On the other hand, the primary approach adopted for *retrieval* (used both in electro–mechanical and computer–based *IR* systems) is called *boolean* retrieval. In *boolean* retrieval, a logical combination of *terms* (commonly referred to *words* in IR literature), which results in a set of documents that exactly match a given query. The alternative approach is to assign a score to each document in a collection reflecting the relevance with respect to a given query. After sorting the documents based on the score the pre–defined number of top documents are the result of a query. Luhn (1957) proposed this method commonly known as *ranked retrieval*. It was tested by Maron et al. (1959) where They used a collection of 200 documents with

manually–assigned keywords and 39 queries demonstrating that it outperformed *Boolean* search. Later Luhn (1958) proposed that,

> The *frequency of word occurrence* in an article furnishes a useful measurement of *word significance*. – Luhn (1958)

This approach later became known as *term frequency weighting*. The *ranked retrieval* approach has been modified over the following decades by the *IR* researchers in terms of the means by which documents were sorted in relation to a query. The advantages of this method over the *Boolean* search has been presented in the book by (Spärck Jones, 1981, pp. 237). These early developments (mostly in the 1950's) allows the consolidation of *IR* as an increasingly significant research area.

2.1.3 The Major Developments upto mid-1990

At the early stages (during the 1960's) there has been more questions asked than answered. *Gerard Salton* [4] was one of the influential people leading a large IR group, first at *Harvard University* and later at *Cornell University*. The group produced many technical reports (the *ISR* reports), establishing ideas and concepts that are still major areas of research today. One of the important concept was *formalization of algorithms to rank documents* for a given query and one notable solution was to view each document as a *N–Dimensional* vector[5]. It was originally proposed by Switzer (1963) and later (Salton, 1968, pp. 236) suggested the *cosine of the angle between two vectors using the cosine coefficient* to measure the similarity between a document and a query vector. *Relevance Feedback* (Rocchio, 1965), a process to support iterative search, where documents previously retrieved could be marked as relevant, is another significant innovation of this period. A modification of this concept is used in modern search engines (e.g. *Related Article* links in *Google Scholar*), where the user query is modified automatically using the information extracted from the relevant documents.

One of the primary focus of our research, *Document Clustering* is also an innovation came into practice during this era. a detailed review of document classification is presented in § 2.1.5. This also includes the pioneering of the use of semantic similarity of documents using semantic variation of the query terms. The books by Salton (1968), van Rijsbergen (1979) and Stevens (1965) can be consulted for the past researches on this topic. The description

[4] https://en.wikipedia.org/wiki/Gerard_salton

[5] N being the number of unique terms (words) in the collection

of one of the earliest commercial systems, also started emerging during this period, can be found in the article by Dennis et al. (1962). According to Bjørner and Ardito (2003), one of the the first data search companies was *Dialog* formed in 1966 due to the creation of an *IR* system for *NASA*. One striking aspect of this period was the extensive use of *Boolean* search especially in the commercial systems, regardless of the consistent demonstration of the superiority of *Ranked Retrieval* by the researchers. It most probably can be contributed to the low level of interaction between researchers and commercial entities (Sanderson and Croft, 2012). Mooers (1961) also criticized this practice by stating that,

> It is a common fallacy, underwritten at this date by the investment of several million dollars in a variety of retrieval hardware, that the algebra of George Boole (1847) is the appropriate formalism for retrieval system design. This view is as widely and uncritically accepted as it is wrong.. – Mooers (1961)

One important innovation and very significant for our research developed during the 1970's, was the use of ***t**erm **f**requency (tf)* weights i.e. term occurrence within a document. It was complemented by the ***i**nverse **d**ocument **f**requency (idf)* i.e. the inversely proportional relation of a term across a collection in terms of retrieval significance (Jones, 1988). Salton and Yang (1973) presented an early investigation on combining the two concepts together (i.e. *tf–idf*), which is now widely used in *IR* research. We have used *tf–idf* extensively in our research, especially in our *ML* experiments thus, more details is presented below.

TF–IDF

The ***T**erm **F**requency – **I**nverse **D**ocument **F**requency (TF–IDF)* metric is very popular in the *IR* and *IE* community, especially for *Topic Modelling* and *Text Summarizing* tasks. The origin of this metric is widely attributed to Jones (1988), originally published in 1972. The proposed *term specificity metric* became ***I**nverse **D**ocument **F**requency (IDF)*. It is the heuristic representation of the intuition that if a *term* occurs in many documents, it is an indication of lack of information content to represent the specificity of any document. The intuition proved to be of significant importance in the growing IR research at that time and continues to be so. ***T**erm **F**requency (TF)* on the other hand measures the frequency of a *term* in a given document and intuitively in this case higher is a better indicator of the importance of the term in relation to the document. Coupled together *TF–IDF* is the basis for many if not most term weighting schema (e.g. BM25) (Robertson and Zaragoza, 2009). The metric has been proven to be extremely robust and often outperforms many carefully designed and knowledge rich models. It is also been in used beyond text based retrieval (i.e. for other media) and general *NLP* (Harman, 2005).

The original literature (Jones, 1988) claimed very little theoretical basis for the intuition other than a reference to some versions of the *Zipf's Law* (Zipf, 1949) concerning term frequencies. It is neither a strong argument nor an accurate one since, the definition of term frequency used in *Zipf's Law* is quite far from the definition in use for IDF i.e. frequency of a term in a continuous body of text for *Zipf's Law* in contrast to that of IDF: how many document contains at least one occurrence of a term (frequency of a term in individual document is irrelevant). Formally, if there are N documents in a collection and *term* t_i occurs in n_i documents[a] and the total frequency of the *term* t_i in a document d_n is f_i^n then the general formula for *TF–IDF* is defined as follows,

$$tf(t_i,\, d_n) = \begin{cases} f_i^n \\ \log(1+f_i^n),\ \text{log normalized version} \end{cases}$$
$$idf(t_i) = \log \frac{N}{n_i}$$
$$tf.idf(t_i,\, d_n) = tf(t_i, d_n) \times idf(t_i)$$

A high *TF–IDF* is produced by high frequency of a *term* in the document d_n while having a low document frequency i.e. the term t_i can be found in a lower number of documents. Since, the $\frac{N}{n_i}$ always produces a value higher than or equal to *1*, *IDF* (and in turn *TF–IDF*) always have a value higher than or equal to *0*. As a term t_i occurs in more and more documents, *IDF* approaches *0*, thus if t_i is found in all documents, the *IDF* becomes 0. The probabilistic model for *IDF* tries to estimate the negative logarithm for the probability of relative document frequency i.e. for a given document d in the document set D of size N, the probability of the presence of the term t in d is,

$$P(\, t \mid d) = \frac{|\{d \in D : t \in d\}|}{N} \qquad \text{and thus}$$
$$idf = -\log P(\, t \mid d)$$

Robertson (2004) argued that the probabilistic interpretation is in turn equivalent to that of the *self–information*[b] and such *Information Theoretic* notions are nevertheless problematic since to satisfy such a notion, the appropriate event spaces for the documents are needed for the required probability distribution.

[a] what can be construed as a *term* is not a concern in the calculation and should be defined based on the implementation and the task.

[b] *self–information* is a measure of information content associated with an event in the probability space or the value of a discrete random variable. Although, sometimes *entropy* is synonymous with *self–information*, however, *entropy* is the expected *self–information* except for some specific condition where they can be the same.

Later, Salton et al. (1975) presented the *vector space model*, another innovation of this era, a synthesis of the work done on *document vectors* by his research group. The algorithm is rarely used nowadays, however, the vector representation of documents is a still a common practice

(Sanderson and Croft, 2012). Another alternative to model *IR* system that emerged during this period was the use of *Probability Theory*. Robertson (1977) defined the principals for the probabilistic ranked retrieval which was presented in the article by Robertson and Spärck Jones (1976). van Rijsbergen (1979) also presented a derivation of the probabilistic model which also made the unrealistic assumption that the occurrence of the *terms* in a document are independent of each other. However, these developments led to wide range of researches in later years. Based on the earlier development, during the 1980's and early 1990's several significant innovations were introduced into main stream *IR* research. Different variations of the *tf–idf* weighting schema that was developed during this period has been extensively discussed by Salton and Buckley (1988).

Furthermore, the original probabilistic retrieval models did not include the *tf* weighing and in an attempt to do so the research led to the development of *BM25* ranking function (better known as *Okapi BM25*) as part of the *Okapi* experimental system (Robertson, 1997). Advancement in the basic vector model was also made in this era, the more well–known is the ***L**atent **S**emantic **I**ndexing (LSI)* where, the effective dimension of the vector space of a collection can be reduced using ***S**ingular **V**alue **D**ecomposition (SVD)* (Deerwester et al., 1990). It was claimed that the reduction merges the *terms* with common semantic meaning thus allows any query to match wider range of documents. It is a purely mathematical approach to the retrieval problem and we have used it in our research extensively. The general class of dimension reduction techniques are commonly known as ***P**rincipal **C**omponent **A**nalysis (PCA)*. However, there are significant advantage of using *SVD* over the classic *PCA* from both mathematical and implementation point of views.

Principal Component Analysis (PCA)

PCA is a statistical procedure to convert a set of observations of probably correlated variables into a set of values of linearly uncorrelated variables called *Principal Components* using an *Orthogonal Transformation*. The number of *Principal Components* are less than or equal to the original number of variables. It was originally developed by Pearson (1901), the method is primarily used for exploratory data analysis and for developing predictive models. PCA has been considered to be one of the most important results from the field of applied linear algebra (Shlens, 2014). Jolliffe (1986) defined PCA as,

> "an *orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on*."

The principal is to decompose a document matrix (D) into the Eigenvector (V) and Eigenvalue (λ) matrix, then the corresponding Eigenvector for the highest Eigenvalue will be the first *Principal Component*, the next highest be the second *Principal Component* and so on. the PCA viewpoint requires the computation of the eigenvalues and eigenvectors of the covariance matrix, which is the product DD^T. Since the covariance matrix is symmetric, the matrix is diagonalizable, and the eigenvectors can be normalized such that they are orthonormal[a].

$$DD^T = V\ \lambda\ V^T$$

Another feature, which is very useful for both data analysis and learning algorithms is *Dimension Reduction*. Since the higher ranked *Principal Components* represent higher variance the first two components already allow researchers to look at a multidimensional data in two dimensional representation and observe useful patterns. Furthermore, reduced dimension representation allows learning algorithms to learn from a more compact search space assuming the claim that *PCA* preserves most of the information even after dimension reduction, is true. However that assumption is made on the basis of another assumption that in the underlying data model (m) actual data (d) and noise (n) are independent i.e. $m = d + n$. Linsker (1988) showed that if the data is *Gaussian* with a covariance matrix proportional to the identity matrix, PCA actually maximizes the mutual information between the original data and the dimensionality reduced output. In our research we have used a different variant of *PCA* i.e. *SDV* for practical reasons and the following subsection will present the theoretical principals of *SVD*.

Singular Value Decomposition (SVD)

Fundamentally, SVD is another way of performing PCA and usually used for achieving similar conclusion as with PCA. It is based on a theorem from liner algebra which suggests that a rectangular matrix e.g. the document matrix D (let us consider the dimension to be $m \times n$) can be decomposed into the product of three matrices: an orthogonal matrix U, a diagonal matrix S, and the transpose of an orthogonal matrix V i.e. V^T. The formal representation can be,

$$D_{mn} = U_{mm}\ S_{mn}\ V^T_{nn};\ \textit{where}\ U^TU = I\ \textit{and}\ V^TV = I$$

The columns of U and V are orthonormal *Eigenvectors* of DD^T and S is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order. If we try to construct the covariance matrix from this decomposition (as in the equation for *PCA*) we have,

$$DD^T = (USV^T)(USV^T)^T$$
$$DD^T = (USV^T)(VSU^T)$$

Since V is a orthogonal matrix, $VV^T = I$ thus,

$$DD^T = US^2V^T$$

So, the square roots of the eigenvalues of DD^T in PCA are the actual values of D in SVD, thus, using the SVD to perform PCA is more convenient numerically than forming the covariance matrix for the original PCA, since the formation of the covariance matrix can cause loss of precision (Jobson and Korkie, 1980).

[a] in linear algebra, two vectors in an inner product space are orthonormal if they are orthogonal and unit vectors. (see, https://en.wikipedia.org/wiki/Orthonormality)

Unlike *LSI*, many linguistic approaches have been attempted in this era with little or no improvement over the existing approaches (Sanderson and Croft, 2012). One approach that showed promise was *Stemming*. Regardless of the existence of stemming algorithms since the 1960's, the stemming rules developed by Porter (1980) continues to shape the stemming development even today. Another change in the perception of *IR* especially in the academic community during the this period was the recognition and eventual development of large scale collection for testing performance to parallel the already a commonplace phenomenon in the industrial practice. ***T**ext **RE**trieval **C**onference (TREC)* is one of the earliest such attempt for an annual exercise where a large number of international research groups collaborated to build test collections several orders of magnitude larger than had been in existence before (Deerwester et al., 1990). The experience showed that the existing weighting and ranking methods are not equally suitable for all types of dataset which eventually was confirmed when web search engines started to get popular in the late 1990's.

During this period the use of ***M**achine **L**earning (ML)* started to be more common as a substitute for the earlier practice of manually devised and tuned by hand *IR* methods. Fuhr (1989) was one of the earliest to adopt such approach, proposing the retrieval function to be learned based on relevant documents identified for an existing set of queries. In contrast to the popular Rocchio's relevance feedback based algorithm (Rocchio, 1971) where a query is tuned for a particular search, Fuhr (1989) attempted to make an adaptive ranking function that can be tuned for a given search space (collection). The idea quickly followed by further research (e.g. Cooper et al., 1992; Fuhr and Buckley, 1991) but, only became effective when large amount of training data became available (e.g. web data) and by using improved learning algorithms and of course hardware those were capable of handling large feature sets.

2.1.4 State–of–The–Art of IR

The real break through in *IR* research (at least in our opinion) came due to the availability of large amount of digitized data, especially web data. Although, until 1993 the number of web pages were quite limited. According to Gray's survey[6,7] the growth since 1993 quadrupled roughly in every *six* months. Web search engines started to appear around the same time and thus, the study of a new set problems in *IR* began. This period also marks the era in which strong interaction was formed between commercial entities and academic groups. The growth of web also pushed forward the development and improvement of web search engines. The early web crawlers traverse the links found in a page and links found by following the first set of links as so on, gathering large amount of web content. However, the first full text search engine using a crawler was *WebCrawler* released in 1994, thus the age of related content search was borne. By analyzing the content the ranking of a web page can be altered thus relevant search results can be produced. The work by McBryan (1994) presented the one of the necessary concepts: *anchor text* (almost always a brief summary of a web page) recognizing it as a valuable resource for the task. *Anchor Text* for example has been a key feature of the *Google* search engine from the very beginning (Brin and Page, 1998). The other significant development was *link analysis* methods such as *PageRank*, developed around the same time (Kleinberg, 1999). One significant drawback of applying *link analysis* and multiple text representations to the existing ranking functions was highly complex algorithms and increasing difficulty in setting accurate parameters for all these features. However, using the logs of user interaction (although quite noisy) from search engines were proposed to revisit the old idea by Fuhr (1989) but with a large training data, e.g. Joachims (2002) used these logs to train a rank function.

Automated exploitation of information extracted from the logs of search engines was also examined in this time. Although not entirely a novel endeavour (e.g. Meister and Sullivan, 1967), the primary use of these logs until this point in time has been exclusively to inform subsequent manual adjustment of a searching system (Robertson and Hancock Beaulieu, 1992). As web user increased, the examination of large amount user query, click pattern and query reformulation extracted from the logs allowed researchers to understand the intention of the user better. The following research directions among many others were somewhat the product of the *web boom*,

- Automated spell correction (e.g. Cucerzan and Brill, 2004)

[6] credit to "Matthew Gray of the Massachusetts Institute of Technology"
[7] http://www.mit.edu/ mkgray/net/web-growth-summary.html

- Automated query expansion (e.g. Radlinski and Joachims, 2005)
- Accurate stemming (e.g. Peng et al., 2007)

Similar to the web query logs, it was recognized by the researchers that the same query can mean very different information need for different users (Verhoeff et al., 1961). *IR* systems only started to address this realization (aptly named *differently relevant documents*) only during the late 1990's. There has been attempts to solve this issue before that and one of the prominent example could be the ***M**aximal **M**arginal **R**elevance (MMR)* based system put forth by Carbonell and Goldstein (1998). Another related development was the use of probabilistic models based on *language modelling* (e.g. Hiemstra, 1998; Ponte and Croft, 1998). The *language modelling* made it possible for the researchers to look at many *IR* processes (e.g. relevance feedback, forming clusters of documents, and term dependence) in a new light. For example, the use of *term dependence* as a proximity operator in a ranking function by Metzler and Croft (2005), demonstrated to outperform term dependence free models. As technology evolves *IR* became almost synonymous to search as it attempts to deal with new problems, e.g. the rapid progression of social media pushed *IR* research to work on social search. An early work by Dumais et al. (2003) on desktop search has a lot of characteristics common to the social media oriented search research, such as, user tagging, conversation retrieval, filtering and recommendation, and collaborative search. Just like social media search, the necessity of complex question answering based search diverted *IR* research from its core of simple user query with little or no linguistic structure, to its current state started with the *Q&A* track in *TREC* (Voorhees, 2001). The state of *IR* research thus can be a benchmark not only of the state of knowledge retrieval but also the information requirement of a particular time period. One significant aspect of *IR* is document classification and due to the importance of this aspect in our research, a detail overview is presented in the next sub–section.

2.1.5 Text Classification

Classification or the task of grouping a set of elements (text, document etc.) into pre–defined groups or classes thus the task is known as classification task. It has been studied extensively in the fields such as database, data mining and *IR*. Formally the problem can be defined as follows, considering we have a set of training elements or instances $D = \{X_1, X_2, \ldots X_N\}$ and each instance is labelled with a class value from a set of k different classes $\{1, 2, \ldots k\}$. Conventionally the training data is used to develop a classification model identifying the

features that allow each instance to be linked with the associated label. The model then can be used to predict the labels of instances for which the label is previously unknown, commonly known as the test instances. On the basis of labelling, classification tasks can be of a hard variant, where labels are explicitly assigned to each instance, or of the soft variant, where, ranking of different class choices for the a test instance is predicted, or the assignment of multiple labels is allowed (Gopal and Yang, 2010).

The problem of text classification, a specific class of the general classification problem tuned towards classifying blocks of text (can be a text blocks or set of documents) is closely related to that of classification of records with set–valued features (Cohen, 1996). However, this model assumes that the presence or the absence of words used in a document convey all the information, reflecting the view posed by the binary model in *IR*. In reality the frequency of words plays a significant role in text classification models and the typical domain–size of text data (the entire lexicon size) is much greater than a typical set-valued classification problem. A detailed survey of classification methods can be found in the book by James (1985) and Duda et al. (2000). A survey specific to the text domain can be found in the article by Sebastiani (2002). Furthermore, a relative evaluation of different text classification methods can be found in the article by Yang and Liu (1999).

Many of the text classification methods have software implementation in the form of publicly available toolkits such as,

- BOW toolkit (McCallum, 1996)
- Mallot (McCallum, 2002)
- WEKA (Hall et al., 2009)
- LingPipe (Alias-i, 2008)

The text classification has been used in a wide variety of domains in *IR*. For example, *News Filtering and Organization*, the process of categorizing news items (e.g. Lang, 1995), *Document Organization and Retrieval*, a similar task but extended to incorporate text sources beyond news sources, especially useful for full text search in web search engines (e.g. Armano, 2015), *Opinion Mining*, the task of identifying the opinion being expressed in a text (e.g. Serrano Guerrero et al., 2015), *Email Classification and Spam Filtering*, the task of classifying e–mails (e.g. Alberts and Forest, 2012) and identifying *junk* e–mails (e.g. Bajaj and Pieprzyk, 2014) etc. A wide variety of methods can also be employed to model the classifier. the set of methods that are generally used also exist for other data domains (e.g. quantitative and categorical data) and applying them for text using frequencies of

word attributes allows most of the methods for quantitative data directly on text. However, the particularity of text data being sparse, often high dimensional and low frequency of many words requires classification models to explicitly consider and account for these characteristics. Some of the commonly used classes of methods are,

- **Pattern (Rule) based Classifiers:** A class of classifiers that uses the word patterns which are most likely to be related to the different classes to model the classifier. A set of rules are usually constructed, in which the left hand side corresponds to a word pattern, and the right hand side corresponds to a class label. These rules are then used for the purposes of classification.
- **Decision Trees:** Decision trees are designed with the use of a hierarchical division of the underlying data space with the use of different text features. The hierarchical division of the data space is designed in order to create class partitions which are more skewed in terms of their class distribution.
- **Bayesian (Generative) Classifiers:** Bayesian classifiers attempt to build a probabilistic classifier based on modelling the underlying word features in different classes. The idea is to classify text based on the posterior probability of the documents belonging to the different classes on the basis of the word presence in the documents.
- **SVM Classifiers:** SVM Classifiers attempt to partition the data space with the use of linear or non–linear delineations between the different classes and determine the optimal boundaries between different classes.
- **Neural Network Classifiers:** Artificial Neural Network based classifiers are related to SVM classifiers (fall in the category of discriminative classifiers) and are in contrast with the generative (Bayesian) classifiers (see, Jebara, 2003)
- **Nearest Neighbour Classifiers:** The vicinity of the instances based on the features is the basis for this type of classifiers. We have used *k–means* algorithm in our research that fall into this category of classifiers.

The general development life–cycle of any classifier starts with two very significant sub–tasks, namely, document representation and feature selection or feature transformation. Text can be represented in two separate ways. The first is as a bag of words, in which a document is represented as a set of words, together with their associated frequency in the document. Such a representation is essentially independent of the sequence of words in the collection. The second method is to represent text directly as strings, in which each document is a sequence of words. Like most text classification methods we also used the bag–of–words representation in our research because of its simplicity for classification purposes. Feature selection on the other hand, is especially important in text classification due to the high dimensionality of text features and the existence of irrelevant (noisy) features. While Feature selection is the process of identifying relevant features from a very large feature set, feature transformation

reduces the dimensionality of the feature set by creating a new lower dimension as a function of the original set of features.

Generally Used Feature Selection Methods

Gini Index:

Let $p_1(w)\ \ldots\ p_k(w)$ be the fraction of class–label presence of the k different classes for the word w. In other words, $p_i(w)$ is the conditional probability that a document belongs to class i, given the fact that it contains the word w. Then, the *gini–index* for the word w, denoted by $G(w)$ is defined as,

$$G(w) = \sum_{i=1}^{k} p_i(w)^2$$

The value of the *gini–index* $G(w)$ always lies in the range $(frac1k, 1)$. Higher values of the $G(w)$ represent a greater discriminative power of the word w. For example, when all documents which contain word w belong to a particular class, the value of $G(w)$ is 1. On the other hand, when documents containing word w are evenly distributed among the k different classes, the value of $G(w)$ is $frac1k$. For a document set containing n documents, d words, and k classes, the complexity of the information gain computation is $O(n \cdot d \cdot k)$. This is because the computation of the term $p_i(w)$ for all the different words and the classes requires $O(n \cdot d \cdot k)$ time.

Information Gain:

A commonly used related measure for text feature selection is *information gain* or *entropy*. Let P_i be the global probability of class i, and $p_i(w)$ be the probability of class i, given that the document contains the word w. Let $F(w)$ be the fraction of the documents containing the word w. The information gain measure $I(w)$ for a given word w is defined as,

$$I(w) = \sum_{i=1}^{k} P_i \cdot \log(P_i) + F(w) \cdot \sum_{i=1}^{k} p_i(w) \cdot \log(p_i(w)) + (1 - F(w)) \cdot \sum_{i=1}^{k} (1 - p_i(w)) \cdot \log(1 - p_i(w))$$

The greater the value of the information gain $I(w)$, the greater the discriminatory power of the word w. For a document set containing n documents and d words, the complexity of the information gain computation is also $O(n \cdot d \cdot k)$.

Mutual Information:

The *mutual information* measure is derived from *information theory* (Cover and Thomas, 1991, see,), and provides a formal way to model the mutual information between the features and the classes. The point wise mutual information $M_i(w)$ between the word w and the class i is defined on the basis of the level of co–occurrence between the class i and word w. The expected co–occurrence of class i and word w on the basis of mutual independence is given by $F(w) \cdot P_i$ and the true co–occurrence

is given by $F(w) \cdot p_i(w)$. The mutual information is defined in terms of the ratio between these two values.

$$M_i(w) = \log(\frac{F(w) \cdot p_i(w)}{F(w) \cdot P_i}) = \log(\frac{p_i(w)}{P_i})$$

The word w is positively correlated to the class i, when $M_i(w) > 0$, and negatively correlated, when $M_i(w) < 0$. We can also compute the overall mutual information as a function of the mutual information of the word w with the different classes. These are defined with the use of the average and maximum values of $M_i(w)$ over the different classes.

$$M_{avg}(w) = \sum_{i=1}^{k} P_i \cdot M_i(w)$$

$$M_{max}(w) = max_i\{M_i(w)\}$$

Either of these measures may be used in order to determine the relevance of the word w. The second measure is particularly useful, when it is more important to determine high levels of positive correlation of the word w with any of the classes.

χ^2–Statistic:

It is a different way to compute the lack of independence between the word w and a particular class i. Let n be the total number of documents in the collection, $p_i(w)$ be the conditional probability of class i for documents which contain w, P_i be the global fraction of documents containing the class i, and $F(w)$ be the global fraction of documents which contain the word w. The χ^2–statistic of w between the word w and class i is defined as,

$$\chi_i^2(w) = \frac{n \cdot F(w)^2 \cdot (p_i(w) - P_i)^2}{F(w) \cdot (1 - F(w)) \cdot P_i \cdot (1 - P_i)}$$

Just like the mutual information, we can compute a global χ^2–statistic from the class–specific values. We can use either the average or maximum values in order to create the composite value.

$$\chi_{avg}^2(w) = \sum_{i=1}^{k} P_i \cdot \chi_i^2(w)$$

$$\chi_{max}^2(w) = max_i\{\chi_i^2(w)\}$$

The χ^2–statistic and mutual information are different ways of measuring the the correlation between terms and categories. One major advantage of the χ^2–statistic over the mutual information measure is that it is a normalized value, and therefore these values are more comparable across terms in the same category.

The feature transformation methods are primarily the *PCA* based approach presented in § 2.1.3. For our research we used transformation rather then selection methods. Two interesting

research questions arose from the standard classification process, since the feature generation and classification modelling are independent tasks.

- Can the feature-specific insights obtained from the intermediate results of some of the classification algorithms be used for creating feature selection methods that can be used more generally by other classification algorithms?
- Do the different feature selection methods work better or worse with different kinds of classifiers?

A detail study on both the questions can be found in the article by Mladenić et al. (2004). They demonstrated, in regard to the first query, that the feature selection derived from *linear classification* provide very effective results. In regard to the second query, they presented that the sophistication of the feature selection process itself was more important than the specific pairing between the feature selection process and the classifier. Text classification is a significant part of our research and this section attempted to provide a general overview of the topic with an introduction to the general practices. We however, used a limited part of the available methods primarily due to the specificity of our data. The next section will give an overview of another significant research topic in the field of data science, *IE*.

2.2 Information Extraction

Traditionally IE is the process of identifying structured information from seemingly free flowing unstructured text written in ***N**atural **L**anguage (NL)*. Digital contents often contain information necessary that is too hard to search directly. However, once the information is extracted in structured (often predefined) classes, information searching becomes as simple as running a query to search data from a database. Piskorski and Yangarber (2013) presented a concise yet insightful definition of IE stating that "IE is an area of natural language processing that deals with finding ***factual*** information in free text". Factual information in this context are structured information such as database records. The objective is to capture real–world entity, events concerning the entities, temporal properties, locative properties etc. embedded in digital text. They (Piskorski and Yangarber, 2013) also argued that searching for specific, targeted factual information constitutes a large proportion of all searching activity on the part of information consumers.

However, IE has not receive as much attention as **I**nformation **R**etrival (IR), even sometimes confused with the latter, most probably due to IR's immediate application possibilities in e–commerce and web search applications. IR is the task to identify a set of documents from

a collection of documents, for the relevance of it to a given query and associate a score for the level of relevance using key–words search and other linguistic measures. The scores are then used to rank the documents. Thus, IR systems usually return a ranked list of documents and usually without any further information on the actual content of those documents. IE systems are not designed to rank, rather extract the meaningful information within the text such as events, entities or relationships to satisfy some predefined knowledge structure. The output then can be used as it is or to infer more complex relationships (e.g. possible trend of an event) by facilitating easier search and retrieval of structured information. In principal IE systems are more knowledge–intensive and harder to develop than IR systems but in practice they can be seen as complementary to each other (Piskorski and Yangarber, 2013). IR systems often use IE systems to extract structure in a document for intelligent ranking whereas, IE systems often use IR systems as pre-filter for a very large document set to reduce the search space by discarding irrelevant documents.

In most cases an IE task is to identify objects of predefined classes in a specific domain ignoring all the irrelevant text and record them in a data structure often referred to as *templates*. For example, if we are interested in extracting information about company mergers or ownership changes, we need to identify entities such as company names and events such as merger, purchase etc. However the process begins with identifying lower level groups in the free–text such as, tokens, words, etc. and then progressively larger groups such as phrases (e.g. noun phrases) and sentences. The levels of complexity of the structures increases as larger groups (e.g. syntactic dependency relations) are formed. IE application also displays somewhat task independent and properly defined hierarchical processing structure upto a certain level. However, it is rather common to have task–specific components in most IE applications. Any IE system can be viewed from the light of the level complexity its intended output contains, i.e. IE represents a set of somewhat similar and interconnected tasks with varying level of complexity e.g. ***N**amed **E**ntity **R**ecognition (NER)* is far less complex a problem than ***E**vent **E**xtraction (EE)* considering EE requires some level of NER in the first place. All these tasks together are considered the IE task family and some of the significant tasks, relevant to this research has been explored later (see §2.4).

According to Appelt (1999) the task of any IE system, from a very generic perspective of course, can be defined in two different forms,

- A description of the types of information the system is looking for, or
- The description of a template

The template description is basically a tabular representation of the extracted information which consist of *attribute–slots* that are to be filled. The IE system will fill each instantiated template with extracted values thus produces *attribute–value* pairs. The value could either be a string directly extracted from the source text or some predefined values inferred by the source text. However, IE is not ***N**atural **L**anguage **U**nderstanding (NLU)*, since the distinct characteristics of IE separate it from NLU or other NLP tasks (Appelt, 1999). He listed the generic characteristics of any IE systems as follows,

- Fixed and often limited domain
- Well established and precise metrics of success
- Fixed and limited representational format

As for the first point, IE tasks are often defined for specific domains. Let us take the task of identifying ingredients from cooking recipes. The objects of interest (i.e. ingredients) thus are restricted to a fairly limited set of lexical items and more restricted in terms of grammatical group in which they belong. Furthermore, the task is clearly defined for a general and again quite restricted domain (i.e. cooking recipes[8]). As a result the events described in the recipe text and the types of relations of interest available thus also be rather limited. Finally, IE tasks have well defined, precise and objective evaluation metric, which is an significant outcome of the historical development of the field and it will be addressed in details later (see §2.3).

Furthermore, most IE tasks are designed to process a large amount of free–running texts, thus the input is expected to be quite noisy (e.g. lexical and syntactic errors are expected to be commonplace). These aspects of the source data increase the processing complexity and processing error significantly. Consequently , IE systems are traditionally implemented with simple finite–state methods for greater processing speed and the use of robust algorithms to increase error–tolerance. Appelt (1999) thus justly described IE as "compromise natural language processing". The domain specific characteristic of IE also implies that it often requires domain specific knowledge and it is usually provided in the form of hand–crafted rules or by using learning algorithms in domain–relevant corpus. Before digging more into design and practices associated with IE systems, some technical terms are needed to be introduced and we are using the definitions provided by the ***N**ational **I**nstitute of **St**andards (NIST)*[9] at the ***M**essage **U**nderstanding **C**onference (MUC)* IE tasks' introduction page.

[8]It is worth mentioning that this particular domain do have an intriguing parallel to the specification description domain and thus one of the test bed for the methods developed for this research

[9]http://www.itl.nist.gov/iaui/894.02/related_projects/muc/

- ***Entity*** is an object of interest such as a person or organization
- ***Attribute*** is a property of an entity such as its name, alias, descriptor, or type
- ***Fact*** is a relationship held between two or more entities
- ***Event*** is an activity or occurrence of interest such as "*a terrorist act*"
- ***Evaluation*** is the assessment of performance according to agreed upon measures

IE research has been at full pace with strong application potential since 1980's and some of the original IE tasks (e.g. NER, ***C****oreference* ***R****esolution (CO)* etc.) even became a part of the standard IE system architecture. However, resonating with Huttunen et al. (2002), we believe that extracting information from free–text written in NL is and has always been a non-trivial task due to the inherent complexity and ambiguity NLs can pose even on a limited domain. Furthermore, a significant portion of the information might even be implicit and thus the requirement of domain–knowledge shall be greater, even off–domain world knowledge may be needed. Then there is the sparse nature of the information distribution, i.e. some search spaces may not contain any information at all, some might contain very few information and some might be saturated with the desired information. However, due to the narrower scope of IE with respect to other NLP tasks that require deeper understanding of text, less sophisticated linguistic analysis tools might be enough to extract relevant information thus implies lower knowledge engineering requirement (Piskorski and Yangarber, 2013).

Piskorski and Yangarber (2013) also argued that due to the advancement of NLP specially the development of efficient, fast, robust and high coverage shallow text processing methods and tools allows IE systems to evade using deep linguistic analysis. It allows the emergence of practical IE methods to be used in in real–world application, capable of analysing large amount of data with reasonable processing time. Furthermore, many of the IE tasks are considered to be core components of other NLP applications such as ***M****achine* ***T****ranslation (MT)*, ***Q****uestion* ***A****nswering (Q&A)*, *Text Summarization*, *Opinion Mining*, etc. Most of the progress can be contributed to the well defined IE task definitions, objective evaluation and the emergence of a well established IE system architecture. An historical development of IE is a good start towards proper understanding of this very interesting research field.

2.3 History of Information Extraction

IE has been and is a part of NLP research domain and the problems it has originally been emerged to solve in one way or another primarily from the need for structured information

which is manageable and easily organizable than the original text that the information are extracted from. Computational IE may be a new technology but it is far from a new idea and one of the earliest linguistic solution for extracting information from free–flowing text was envisioned by Zellig S. Harris (Harris, 1958). He suggested that the co-occurrence based syntactic patterns can be used for such tasks. Although, he hypothesized that the usefulness of this approach can be exploited on a limited number of domains[10]. He presented the concept of syntactic patterns containing similar meaning i.e. "Transformations" and key information holders i.e. "Kernels" and how they can be extracted. Thus, the kernel can be used for direct information search. One can find works with titles like "Text searching with templates" (Wilks, 1987) but those were new ideas and not being backed by significant computational power to execute them (Wilks, 1997). However, what Zellig S. Harris envisioned more than 50 years ago, is yet to be a reality regardless of the recent advances in computational capability, IE methods and NLP technologies. The following sub–sections will attempt provide a concise history of IE research.

2.3.1 Early Works

The generic characteristics of early IE systems were undoubtedly the use of templates as slot–fillers and linguistically motivated rules to extract information. Some of the earlier work has been summarized by Cowie and Lehnert (1996) and he described an IE system presented in Sager (1981), which has been applied to highly domain–specific radiology reports and hospital discharge summary to extract information into traditional ***Co**nference on **Da**ta **Sy**stems **L**anguages (CODASYL)* database management system (Taylor and Frank, 1976). The system used a parsing program to obtain the syntactic relations among sentence words (the fundamental structure of language–borne information) first described in (Sager, 1960). The system was developed under the ***L**inguistic **S**tring **P**roject (LSP)*[11] at New York University that continued between 1960 and 2005, directed by Naomi Sager. The original research focused primarily on the development of a large–scale computational grammar of English (Sager, 1960), the potential application was deriving (more in the direction of inducing) what Sager called *information formats* a structured table–like format which was essentially *templates*. Table 2.1 is an example of a modern *template*.

The vision was to driving away from the abstractions of NL forms and produce a database to extract *facts* and one can easily identify the IE written all over the idea. One important aspect

[10]The focus was scientific articles since they use a rather limited number of syntactic patterns

[11]Sponsored by the American Medical Association

Table 2.1. A filled template from MUC-3 for terrorist domain (Grishman and Sundheim, 1995)

0.	MESSAGE ID	TST1-MUC3-0080
1.	TEMPLATE ID	1
2.	DATE OF INCIDENT	03 APR 90
3.	TYPE OF INCIDENT	KIDNAPPING
4.	CATEGORY OF INCIDENT	TERRORIST ACT
5.	PERPETRATOR: ID OF INDIV(S)	‘‘THREE HEAVILY ARMED MEN’’
6.	PERPETRATOR: ID OF ORG(S)	‘‘THE EXTRADITABLES’’
7.	PERPETRATOR: CONFIDENCE	CLAIMED OR ADMITTED: ‘‘THE EXTRADITABLES’’
8.	PHYSICAL TARGET: ID(S)	*
9.	PHYSICAL TARGET: TOTAL NUM	*
10.	PHYSICAL TARGET: TYPE(S)	*
11.	HUMAN TARGET: ID(S)	‘‘FEDERICO ESTRADA VELEZ’’ (‘‘LIBERAL SENATOR’’)
12.	HUMAN TARGET: TOTAL NUM	1
13.	HUMAN TARGET: TYPE(S)	GOVERNMENT OFFICIAL: ‘‘FEDERICO ESTRADA VELEZ’’
14.	TARGET: FOREIGN NATION(S)	-
15.	INSTRUMENT: TYPE(S)	*
16.	LOCATION OF INCIDENT	COLOMBIA: MEDELLIN (CITY)
17.	EFFECT ON PHYSICAL TARGET(S)	*
18.	EFFECT ON HUMAN TARGET(S)	-

of this work is that the information formats (i.e. templates) in contrast to modern idea of IE were not predefined a priori by experts in the field. The methodology instead, given a set of text in a sub–language domain, induces the information format by performing distributional analysis to identify the word classes in the domain. Gaizauskas and Wilks (1997) presented as an example that, *film shows clouding*, *x-rays indicate metastasis*, etc. allow the system to define the following format,

[TEST | SHOW | MEDICAL FINDING]

Evaluation upto a limited capacity was also carried out by contrasting the program’s behaviour with the results of human experts to fill in a comparable information format only from the information in the discharge summary. However, inducing templates was abandoned through the 1980’s and early 90’s due to the complexity and difficulty, and the use of predefined, domain expert tailored templates had been adopted instead (Gaizauskas and Wilks, 1997).

Another long term project with the primary focus in language understanding, in particular on story comprehension, by Roger Schank and his colleagues (Schank, 1975; Schank and Colby, 1973; Schank and Abelson, 1977) at Yale University. The work was based around the idea that stories follow a limited set of stereotypical patterns which was referred to as *script* by

Schank. He proposed that knowing the scripts, not only for just stories but also for different domains (e.g. a corporate merger, or a management succession event, or a doctor–patient examination session all have predictable role-players and sub-events Gaizauskas and Wilks (1997)), will allow one to make sense of a text describing an instance of such events. The first attempt to build a computational IE system based on the idea was *FRUMP*, implemented by Gerald deJong (DeJong, 1979, 1982). What Cowie and Lehnert (1996) called an attempt to build an explicit IE system, was in fact, a general NLP system to analyze news stories from a *UPI*[12] news–wire feed to generate summary for the users of the system. The system is a *strong reminiscent* of the modern IE systems (Patwardhan, 2010) because of its functional structure to generate summaries is fundamentally a sort of event template filler that outputs a single sentence summary of an event. The system was reported to identify role–fillers for 60 types of situations (Gaizauskas and Wilks, 1997). Hand–coded rules were used to fill a data structure that deJong called *sketchy script* (a simplified variation of *script* (Cullingford, 1978; Schank and Abelson, 1977)), was used to represent events or real-world situations. The system relied on an alternation of *prediction* and *substantiation* modules which used, respectively, top-down, expectation-driven processing relying on predictions from the script and bottom-up, data-driven processing based on input from the text. The work is also notable for carrying out a reasonably extensive evaluation: six days of previously unseen news stories were fed in real–time through FRUMP and the results classified as to whether the stories were processed correctly, nearly correctly, wrongly, or were missed (Gaizauskas and Wilks, 1997).

As an eventuality, the first commercial IE system was developed during the 1980's. The first such system was deployed (to the best of our knowledge) was an automatic system to process money transfer messages between banks, called ATRANS (Lytinen and Gershman, 1986). ATRANS adopted Schank's *script* style approach to process the input text. The script–driven prediction was used to identify actors (e.g. originating customer, originating bank, receiving bank, etc.) in order to fill in a pre–designed template (after human verification) to initiate automatic money transfer. Later, the Carnegie Group developed and deployed, what they then called *fact extraction system* for *Reuter*[13] named JASPER (Andersen et al., 1992). JASPER was designed to process company press releases from *PR Newswire*[14] and fill in a template designed for information about company earnings and dividends. The instantiated templates then were used to produce potential news stories that was then validated or post–edited by

[12] http://www.upi.com/
[13] http://www.reuters.com/
[14] http://www.prnewswire.com/

journalists, reducing story preparation time significantly. It extracted information from small sentence fragments using robust NLP methods (Piskorski and Yangarber, 2013). Another commercial system developed by *GE*[15] during the late 1980's was the SCISOR system (Jacobs and Rau, 1990) which has been used in their financial decision making system called MARS (Bonissone and Dutta, 1990) for the analysis and extract information on corporate merger and acquisition from online news.

There are at least two significant Prolog–based IE systems were developed during this period. The first one was developed by Silva and Dwiggins (1980) that extracts information about satellite–flights from text. The primary object as the authors presented was to update data with the extracted information from natural text. The other system was developed by James Cowie (1983) to extract regularised descriptions of plants (i.e. templates) from field–guide descriptions of plants and animals. The system segments input text into small chunks according to focal points (e.g. pronouns, conjunction, punctuation etc.), thus evades the necessity and consequently the complexity of parsing the text. The approach relied on a hand–crafted domain–specific lexicon of keywords that allowed segments of source text to be compared to the sections of the target templates. Hand-crafted rules associated to slots in the template (properties of a plants) were then connected to the selected text and the extracted property value.

Another significant academic project was the work by Gian Piero Zarri (1983) from the same time period, who described a system that identifies information about relationships and meetings of French historical personalities and extract them in the *ReSeDA*[16] semantic metalanguage. He describes the attempt to translate automatically French texts dealing with biographies of historical figures of the late Middle Ages in France into the *metalanguage* which captured certain semantic relations. He described that the most important characteristic of the *ReSeDA* system (that encompasses the metalanguage architecture) lies in the possibility of using inference procedures to question the database about causal relationships which may exist between the different recorded facts, and which are not explicitly declared at the time of data entry (Léon et al., 1982). The metalanguage itself was organized round case frames for predicates, which can be viewed as small-scale templates: what was to be extracted were the roles in particular historical events, such as the naming to a position of an historical figure by a given body on a particular date at some location (Gaizauskas and Wilks, 1997). The approach starts with syntactic analysis of the text and then performs a semantic parsing during which lexical triggers (i.e. keywords in the domain) cause one or more case frames

[15] http://www.ge.com/

[16] http://w3.avignon.inra.fr/reseda/presentation/index.html

for the primary predicator to be invoked and instantiated using associated text using rules associated to the case frame.

All the above systems and other early IE systems were developed using the ***K**nowledge **E**ngineering (KE)* (Appelt, 1999). The creation of domain knowledge in this approach is often in the form of rules or patterns to detect and extract the target information and usually done by a human domain experts. The patterns or the rules are usually learned by through analysis of test corpus and up to a certain extant intuition. It is an iterative process, i.e. starting with a small set of extraction rules which are tested on the available corpora and extended until a desired trade-off between precision and recall is reached (Gaizauskas and Wilks, 1997). The main difference between IE research in 1990's and the previous decades is the large amount of time and energy that has been spent in the later case, to collect relevant documents, analyse them to produce templates and test corpus. However, most early IE had serious shortcomings: they exhibited a non-modular black-box character, were mono-lingual, were not easily adaptable to new scenarios (Piskorski and Yangarber, 2013) and often had limited and less objective evaluation possibilities specially in terms of comparing one system to another. They however, also demonstrate that relatively simple NLP techniques may be sufficient enough to deal with many real world IE problems. The emergence of proper task definition through influential evaluation campaigns shaped what has been the modern IE research.

2.3.2 History of the Evaluation - MUC

One notable characteristic of IE research is the degree to which its research has been driven by a series of US government–sponsored evaluations. During the mid–1980's several projects sponsored by the US Navy were working on IE from naval communication messages and they felt the need to understand and compare the systems developed under these projects. A number of these ***M**essage **U**nderstanding (MU)* projects decided to work on a set of common message and compare the performance of their respective systems by checking the performance on previously unseen messages. This collaborative attempt was the first of what has turned into a series of key events that modernize and shaped the field of IE that we came to know today. Supported by ***D**efense **A**dvanced **R**esearch **P**rojects **A**gency (DARPA)*, MUCs were organized by the ***N**aval **O**cean **S**ystems **C**enter (NOSC)*. Because of the origin the subject domain of these conferences was defence–oriented such as analysing military messages, searching newspapers for terrorist activities etc. There has been 7 MUCs,

between 1987 and 1998 and except for MUC–1, all provided prepared training–corpus and templates and a structured task definition. Each participant then adapt their system to the given scenario by using the training corpus. Shortly before the conference, participants received a test–corpus and used their systems to fill the provided templates. The results then were sent to the MUC organizer, which had created templates with right answer–keys manually and used them to evaluate the performance. Grishman and Sundheim (1996) provided a detailed summary and findings of the MUC assemblies presented in Appendix G. The MUCs in a way forced and shaped IE technology to the modern notion of the field. Through the process IE tasks and their goals have been defined and objective evaluation of the tasks took shape. It specifically defines each of the subtasks that now defines IE as a discipline, Table 2.2 summarizes the introduction of task definitions in the MUCs,

Table 2.2. Task Definition Development in The Course of The MUCs

	Scenario Template	Named Entity	Template Element	Co-reference	Template Relation	Multilingual Entity
MUC–2	YES					
MUC–3	YES					
MUC–4	YES					
MUC–5	YES					
MUC–6	YES	YES	YES	YES		
MUC–7	YES	YES	YES	YES	YES	YES

2.3.3 Other Evaluation Campaigns

The ***A**utomatic **C**ontent **E**xtraction (ACE) Program*[17] (Doddington et al., 2004), organized annually between 1999 and 2008 (except for 2006) was the next significant evaluation campaign and in some sense an effort to further advance the MUC objectives. LDC developed annotation guidelines, corpora and other linguistic resources for the evaluation and somewhat in the same manner as in the late MUCs. Additional efforts were put into preparing data for languages other than English (e.g. Chinese, Spanish, Arabic etc.). In later edition, they also prepared data for advance tasks such as, global entity detection and recognition that requires cross–document co-reference resolution. The ACE programme primarily defined newer and harder tasks focusing on extracting entities, relations and events throughout the

[17] http://www.itl.nist.gov/iad/mig/tests/ace/

campaign. They achieved this by, including various information sources (e.g., news, web–logs, newsgroups), using high quality input data from sources like telephone conversation transcripts, introducing more fine-grained entity types (e.g., facilities, geo–political entities, etc. instead of simple location), template structures and relation types, and widening the scope of the core IE tasks. Throughout the ACE campaign 3 tasks in IE were focused on,

1. **Entity Recognition**: Renamed to ***E**ntity **D**etection and **T**racking (EDT)*, the task not only deals with names but also all possible mentions (e.g. nominal, pronominal etc.), thus effectively required co-reference resolution. (see § 2.4.1 & 2.4.2)
2. **Relation Recognition**: Named as ***R**elation **D**etection and **C**haracterization (RDC)*, the task was defined to identify and classify relations between entities. There were five general types of relations (e.g. role, part, at, near and social), some of which had been farther divided into subtypes thus totalling 24 types/subtypes of relations. (see § 2.4.3)
3. **Event Extraction**: The task was defined to extract events from a given text and only included in the ACE since 2005 and the 2005 edition had 8 types of events with 33 subtypes. (see § 2.4.4)

Under DARPA led ***T**ranslingual **I**nformation **D**etection **E**xtraction and **S**ummarization (TIDES)*[18] Programs, the LDC at the University of Pennsylvania developed annotation guidelines, corpora and other linguistic resources for the ACE. LDC's ACE annotators tag broadcast transcripts, news–wire and newspaper data in English, Chinese and Arabic, producing both training and test data for common research task evaluations. Both MUC and ACE initiatives are of central importance to IE research, since they provided a set of corpora that are available to the research community for the evaluation of IE systems and approaches.

Both MUC and ACE however, focused on extracting information from a single document, and this aspect of the evaluation has been shifted with the later series of evaluation. Following the new focus, for the classic task of NER, given a name, the extraction system is expected to identify all the mentions and name variations in a large collection of documents. A prime example is the ***K**nowledge **B**ase Population (KBP)*, started as a part of the ***D**epartment **o**f **D**efence (DoD)* supported NIST ***T**ext **A**nalysis **C**onferences (TAC)* from 2009. The primary objective was to bridge the gap between IE and Q&A communities and to promote research in discovering facts about entities and expanding a knowledge base with these facts. KBP has been done through two separate subtasks, Entity Linking and Slot Filling: in 2010, 23 teams submitted results for one or both subtasks.

IE systems and approaches are evaluated in other evaluation campaigns with somewhat broader target research areas. The ***Co**nference on Computational **N**atural **L**anguage **L**earning*

[18] http://metadata.sims.berkeley.edu/GrantSupported/tides.html

(CoNLL) has organized some *shared tasks* on language-independent NER. Some IE-related tasks have been evaluated in the context of the *Senseval* initiative, with the original focus of evaluating semantic analysis systems. For instance, the ***W**eb **P**eople **S**earch (WPS)* task focused on grouping web pages referring to the same person, and extracting important attributes for each of the persons sharing the same name.

2.4 Information Extraction Tasks

Early IE research often took the objectives of a specific problem head on, thus, the signs of the lack of systematic and tasks independent approaches were prominent. One of the earlier Prolog based system was developed by Silva and Dwiggins (1980) and it was found to be rather limited in its application since the system was restricted to single sentences processing and lacked a methodology for extracting complete event description[19]. However, in the late 80's and 90's especially due to the MUC's, IE sub–components are well defined and a standard in most modern IE application. These tasks in their own rights are significantly complex and well defined problems. This section will try to shade some light on some of the significant tasks.

2.4.1 Named Entity Recognition (NER)

NER addresses the problem of identifying a class of predefined entities, namely person (e.g. Mr. Crowley, John Smith, Tom etc.), place names (e.g. The Loire Valley, Paris, The Baltic Sea etc.), Organizations (LIMSI, European Parliament etc.), temporal expressions (e.g. 4th of July, June 22nd, 2015 etc.) and many more. Most of these classes can have sub–classes depending on the level of granularity one sought for a specific application. It has been introduced as a separate evaluation task during the 6^{th} MUC (Grishman and Sundheim, 1996). Some of the sub–tasks e.g. temporal expressions has been found to be very complex a task in itself, thus, since the ACE 2004 (Doddington et al., 2004), there has been a separate task identified and called ***T**emporal **E**xpression **R**ecognition and **N**ormalisation (TERN)*. Also known as ***Timex***, the evaluation for this task is now evaluated in two major temporal annotation challenges: ***TempEval***, now TempEval–2[20] and ***i2b2***.

[19] Multi–sentential processing is absolutely necessary for complete event extraction

[20] http://timeml.org/tempeval2/

NER systems may also extract additional descriptor to fill a small–scale template for the original class element e.g. along with extracting a name the system may also find the title, sex, nationality etc. NER systems may also normalize the entities, especially for highly inflective languages. For example, the name *"Vygintas"* has a number of possible inflected forms in Lithuanian as listed in Table 2.3,

Table 2.3. Inflicted forms of ***"Vygintas"*** in Lithuanian

Form	Case	Example (LT)	Translation (EN)
Vygintas	Nominative	***Vygintas*** yra mano tėtis.	Vygintas is my father.
Vyginto	Genitive	Aš esu ***Vyginto*** dukra.	I am daughter of Vygintas.
Vygintui	Dative	Nupirkau dovanų ***Vygintui***.	I bought gifts for Vygintas.
Vygintą	Accusative	Aš mačiau ***Vygintą*** vakar.	I saw Vygintas yesterday.
Vygintu	Instrumental	Aš nepasitikiu ***Vygintu***.	I do not trust Vygintas.
Vyginte	Locative	***Vyginte*** yra daug blogio.	There is a lot of evil in Vygintas.
Vyginte	Vocative	***Vyginte***, ateik čia!	Come here, Vygintas.

It was during the development of the systems for the early MUC's (3, 4 and 5), researchers started to understand the significance of identifying and classifying names in text. Names are very common in text especially in some domains such as "news" text. Originally three classes were selected for evaluation, person, organization and location. It has since been a rather popular task and evolved in many dimensions. One such dimension was multi-lingual NER, originally at ***M**ultilingual **E**ntity **T**ask Conference (MET)*[21] [22] [23], as a part of DARPA led TIPSTER Text Program and later in ***C**onference **o**n **N**atural **L**anguage **L**earning (CoNLL)* evaluation tasks (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003). Another important dimension being the general recognition of NER as a necessary pre–processing step not only in IE but also in other NLP applications such as ***M**achine **T**ranslation (MT)* and ***Q**uestion **A**nswering (QA)*. The use of NER in Q&A also motivated the development of fine grain NE categories and thus the NE set (Sekine and Nobata, 2004), a hierarchical arrangement of between 100 and 200 named entity categories, came to be. Usually the NE sets are designed for generic use (often for news domain), but many specific sets exist for very specific domains such as biomedical domain, namely for genes and protein name extraction tasks.

[21] MET-1 in 1995 for Chinese, Japanese, and Spanish
[22] MET-2 in 1998 for Chinese and Japanese
[23] http://www.itl.nist.gov/iaui/894.02/related_projects/tipster/met.htm

NER evaluation has also been standardised due to extensive research and evaluation tasks. It was rather complicated at the beginning since the clearest guideline can fall short considering the complexity in NE in free-running text. Even with the three base categories ambiguous situation such as is "Pluto" a place or is "Peter Pan" a person, can arise. Moreover, name polysemy, names that are also places (e.g. "I'll meet you in front of MacDonald's") or places that also represent the governments (e.g. "France and Germany signed a treaty") are rather problematic situations. To deal with the second case the ACE 2008 Evaluation Plan (ACE08)[24] introduces a new category GPE (Geo-Political Entity) for cases like "'France" or "Germany" and reserved location for landmarks such as "Mount Kilimanjaro".

There is also the consideration of selecting and balancing the test and training corpora. If these two are too similar, it gives an unfair advantage to the supervised learning systems. It is also important to put careful consideration about the source of the text and the topic. If the annotators are not familiar with the topic annotation will suffer as a consequence. On the other hand it would be tough for a system to match the performance of an annotator with strong familiarity with the topic. Nevertheless, the following schema is usually used for the evaluation process,

$$Recall = \frac{correct}{correct + missing}$$

$$Precision = \frac{correct}{correct + spurious}$$

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

correct is the tags that agree in extent and type
missing is a tag in the reference corpus with no matching tag in the system output
spurious is a tag in the system output with no matching tag in the reference corpus

The original MUC evaluation had a relaxed scoring method giving partial score if the extent is matched, even though the type is a mismatch. The CoNLL schema does not give any partial score and considered to be the standard for modern NER evaluation scoring. The best F-scores for four languages from the CoNLL multi–site evaluation (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003) is summarized in Table 2.4,

These performances have been demonstrated to plummet sharply if there is a difference in the training and test corpus (Ciaramita and Altun, 2005). They showed that systems trained

[24] http://www.itl.nist.gov/iad/mig/tests/ace/2008/doc/ace08-evalplan.v1.2d.pdf

Table 2.4. Best F-Scores from CoNLL evaluation 2002 and 2003

	Languages			
	English	**Spanish**	**Dutch**	**German**
Best F-Score	88%	81%	77%	72%

with a subset of CoNLL 2003[25] Reuters dataset, managed F-score upto 91% on a test corpus from the same dataset. The score went down to 64% when tested on a Wall Street Journal dataset.

2.4.2 Coreference Resolution (CO)

Among various definition put forward over time I found myself agreeing with Mitkov (1999) that the classical definition by Halliday and Hasan (1976) perhaps is a fundamental yet very solid definition of coreference. The definition is based on the notion of cohesion; anaphora is a cohesion that points back to some previous item. The idea of pointing back is called ***anaphora*** and the entity it refers to is called ***antecedent***. NER is just the first step in identifying ***N**oun **P**hrases (NPs)* over the span of one or more documents. In free–text NPs can be representative of real world entities (also known as ***mentions*** or ***anaphoric noun phrases***) and these entities can be referred in the text as name mention or nominal mention or pronominal mention (Luo, 2007). An example is presented in Table 2.5.

Table 2.5. Possible Mentions of ***Université Paris-Sud*** in Text

Type	**Examples**
Name Mention	Université Paris-Sud
	Université Paris XI
Nominal Mention	The university in Orsay
	The university next to LIMSI
Pronominal Mention	it
	that

The goal of ***A**naphora **R**esolution (AR)* or, as it is also been known since the MUC initiatives, Coreference Resolution[26] (Poesio et al., 2011), is to identify all the mentions of a real world

[25] http://www.cnts.ua.ac.be/conll2003/ner/

[26] see van Deemter and Kibble (2000) for more on the difference between AR and CO

entity in a given text (Ng, 2008). CO thus can be defined as the the task of identifying which parts of a text refer to the same ***discourse*** entity (Poesio et al., 2011). It is rather simple in most cases for a human given the necessary world knowledge. It however, is a rather difficult task for a computer to do so, primarily because of the complexity associated to encode the world knowledge and the nature of complexity this task can pose. Poesio et al. (2011) put together a detail analysis of linguistic and psycholinguistic analysis of the anaphora and anaphora resolution in natural language. They also put an impressive summary of early knowledge based and later data driven computational models for CO.

Annotated corpus became abundant since the MUC conferences and ACE evaluations and prompted the development of supervised ML approaches (Ng, 2010). Early methods, however, similar to many NLP applications were built with hand-crafted rules and oriented towards inference based methods [27]. One of the earliest attempts to resolve anaphora by a computer program is presented in ***STUDENT*** (Bobrow, 1964), a high-school algebra word problem answering system. The ***SHRDLU*** system (Winograd, 1972) and the CO subsystem of the English to French translation system by Wilks (1973) were based on similar methods. These systems heavily relied on inference and lexical clues for the identification of coreferences.

Syntax–based methods are often used to filter unacceptable candidates rather than finding antecedents and plays important role in knowledge–oriented CO systems. Hobbs (1976) presented a syntax–based algorithm, now known as the ***"Hobbs' Naïve Algorithm"***, that is by far the best such algorithm (Mitkov, 1999). Hobbs algorithm is based on various syntactic constraints on pronominal constructions, which are used to search the parse–tree. The optimal search usually ends on a probable antecedent NP for the pronoun from which the search started. He reported that the algorithm works **88%** of the time and goes upto **92%** when augmented with simple selectional constraints. He tested the algorithm on texts from an archaeology book, an Arthur Hailey[28] novel and a copy of Newsweek. Moreover, Hobbs argued that his algorithm is computationally cheap compared to any semantic method for pronoun resolution.

Denis and Baldridge (2007) presented that early data driven learning algorithms based on statistical approaches considered CO as a binary–classification problem, i.e. two mentions from a text $\boldsymbol{x}$ and $\boldsymbol{y}$ have coreferential outcome can be calculated by estimating, $P_c(COREF|\langle x,y\rangle)$. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are ordered pairs, i.e. $\boldsymbol{x}$ is an antecedent preceding in the text for the anaphora $\boldsymbol{y}$, which is an asymmetric interpretation of the problem (Nguyen and Kim, 2008). This

[27] see Hirst (1981) for a detail summary of early coreference resolution systems
[28] http://en.wikipedia.org/wiki/Arthur_Hailey

interpretation of CO is very similar to AR, where one tries to find the antecedent ***x*** of a pronominal ***y*** (Clark and González Brenes, 2008). Many of the reviewed articles called it a ***pairwise model*** (Ng, 2008; Ng and Cardie, 2002; Soon et al., 2001; Yang et al., 2003), and furthermore, Ng (2008) summarized the later shifting of the focus in terms of methods from pairwise models towards other directions as presented in Table 2.6,

Table 2.6. Switch in Research Direction from Pairwise Resolution

Method	Reference
Rich Linguistic Features	(Ji et al., 2005; Ponzetto and Strube, 2006)
Joint Learning	(Daumé III and Marcu, 2005)
Joint Inference	(Denis and Baldridge, 2007)

However, Ng (2008) also argued that the use of rich feature oriented methods along with complex models for CO made these approaches heavily depend on data and thus rather difficult even impossible to use for languages with little or no annotated data. The growing need of multi–lingual CO leads to the development of weakly supervised approaches for automatic processing of low–resource languages. Some of such algorithms are self–training and co–training (Blum and Mitchell, 1998) and ***E**xpectation **M**aximization (EM)* (Dempster et al., 1977) that has been applied to CO (Ng and Cardie, 2003). These methods use a small amount of annotated data with a large amount of unlabelled data and usually incrementally augment labelled data by iteratively training a classifier to label the unlabelled data.

The other aspect of the research on CO, just like any other NLP application research is the objective evaluation of the CO systems. Early systems were often evaluated on small evaluation corpses and until the MUC initiative, organized large–scale evaluation with proper task definition, were rare if not totally absent. The ACE also focused on CO evaluation upto certain extent i.e. the participants only had to extract certain types of relations between predefined set of entities. In contrast to MUC and ACE, the ***A**naphora **R**esolution **E**xercise (ARE)* (Orăsan et al., 2008), in terms of long term goals, focuses on identifying linguistically motivated large set of relation between many types of entities in multiple languages. There are four separate tasks in ARE, namely,

- ***Task 1:*** Pronominal anaphora resolution on pre-annotated texts
- ***Task 2:*** Co–referential chains resolution on pre-annotated texts
- ***Task 3:*** Pronominal anaphora resolution on raw texts
- ***Task 4:*** Co–referential chains resolution on raw texts

Tasks 1 and 2 evaluate the resolution algorithms on an almost perfect input (i.e. input in which the entities to be resolved are known). On the other hand Tasks 3 and 4 simulate application oriented situations where there is no guarantee that the entities to be resolved can be correctly identified. Tasks 1 and 3 focus on pronominal anaphora resolution and require that for each referential pronoun an antecedent is determined, whilst Tasks 2 and 4 address the problem of coreference resolution where entities that refer to the same thing in real world need to be clustered together.

2.4.3 Relation Extraction (RE)

The objective of the task of *Relation Extraction* is to recognize the assertion of a particular relationship between two or more entities in text (Banko and Etzioni, 2008). The task is to predict a relation from a set of predefined relations usually between a pair of entities (defined as a binary relation). The following sentences can be examples of potential relation extraction scenario,

Example 2.1 *Munshi* ***works*** *at LIMSI*

Example 2.2 *Munshi* ***lives*** *in France*

Example 2.3 *Munshi is* ***from*** *Bangladesh*

These are rather simple examples and each sentence is representing one binary relation, in these cases between the entity "***Munshi***" and some other entity in each sentence [29]. However, there are also research in the direction of beyond binary relations i.e. more than two entities representing a relation, also known as ***n–ary*** relations. An instance in a n–ary relation is a list of entities $\{e_1, e_2, e_3, ...e_n\}$ where e_i is entity type. Bach and Badaskar (2007) presented the following example to illustrate the scenario,

"*If we are interested in the ternary relation* **(organizer, conference, location)** *that relates an* **organizer** *to a* **conference** *at a particular* **location**. *For the sentence [***ACL-2010 will be hosted by CMU in Pittsburgh***], the system should extract* **(CMU, ACL-2010, Pittsburgh)**."

McDonald et al. (2005) proposed a framework for extracting 4–ary relations from biomedical abstract text. The system uses the existing methods and systems to extract all the instances of binary relations in the first pass. Those instances were then used to cluster the desired, more complex relations. n-ary predictions are more complicated and considered to be a different

[29] In Example ***2.1*** the relation can be "***employment-related***" and between "***Munshi***" and "***LIMSI***"

IE task all together, namely ***event detection*** is generally considered to be a n-ary RE task. Relation detection and extraction has been introduced as separate task in MUC-7[30] and the relations task covered three relations involving organizations,

- ***location_of***
- ***employee_of***
- ***product_of***

Relation extraction was a significant part of the ACE[31] evaluation, which has been described as ***R**elation **D**etection and **C**haracterization*, introduced in 2002 and revised repeatedly to create a set of relations that can be annotated consistently. Many of the research has been done using the task definitions from 2003, 2004 and to a lesser extent 2005. Each task defined a set of relation types and subtypes, for example, the ACE 2004 relation task has the following tasks and subtasks[32] presented in Table 2.7,

Table 2.7. ACE Evaluation 2004 Relation Task Summary

Relation Type	**Subtypes**
physical	located, near, part–whole
personal–social	business, family, other
employment/membership/subsidiary	employ–executive, employ–staff, employ–undetermined, member–of–group, partner, subsidiary, other
agent–artifact	user–or–owner, inventor–or–manufacturer, other
person–org affiliation	ethnic, ideology, other
GPE affiliation	citizen–or–resident, based–in, other
Discourse	–

However, ACE guidelines explicitly distinguish between reference and reference mention, maintaining the distinction between entity and entity mentions (§2.4.2). Consider the sentence "*His brother came over.*", so the relation mention will record "*His*" and "*His brother*" having a "*family–type*" relation. One has to look at a lookup table to extract the actual entities and will require anaphora resolution. Moreover, ACE relation task guidelines require both arguments of a relation to appear explicitly in a sentence, a constraint shared by many systems that has been developed for the task. Thus, following the guidelines a relation (e.g. employ–staff between "*Munshi*" and "*LIMSI*") cannot be extracted from the sentence,

[30] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/overview.html
[31] https://www.ldc.upenn.edu/collaborations/past-projects/ace
[32] https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-rdc-v4.3.2.PDF

There are 60 PhD students working at LIMSI. Munshi joined about 3 years ago.

The ***T**ext **A**nalysis **C**onference – **K**nowledge **B**ase **P**opulation (TAC-KBP)*[33] evaluations have a sort of RE task called ***Entity Linking***. The task is to extract a set of attributes for a named person or organization. These attributes often have values that themselves are entities. Unlike ACE however, the entities for a relations are not required to be explicitly mentioned in the same sentence. Ji and Grishman (2011) reported that for about 15% of the attributes, cross-sentence analysis is required.

The evaluation of relation extraction is rather straight forward and similar to NER (§2.4.1) considering of course the scores depend on the corpus and the set of relation types. There are other factors such as, entities may be hand crafted or automatically extracted. In the case of the later, rate of error is much higher. There is also the fact that, many research reports their results on just the type of relation extracted and not on the specific subtypes. Specially in ACE evaluation relation and relation mention affect the result considering the annotation of the reference data. The best system in ACE achieved f-measure of 75% using hand–identified entities and classifying to relation type level only. performance goes down to 40% if system–produced entities are used.

2.4.4 Event Extraction (EE)

Event extraction as mentioned earlier (§2.4.3) can be considered to be a more general problem of prediction involving arbitrary number of argument i.e. n-ary relation extraction. However, we are not considering event extraction involving documents containing a single event (e.g. seminar announcements) that are known to describe a known event[34]. There is a whole spectrum of complexity of information structure that can be extracted from text. At one end of this spectrum are elementary events which may take 2 or 3 primary arguments and some optional modifiers (e.g. time, place etc.). ACE 2005 event extraction task includes 33 such event types as listed in Table 2.8,

On the other end of that spectrum is extracting events from a document where the information is sparsely distributed. For example, extracting information about a disease outbreak from news article is fundamentally very different from extracting information about a conference from the call for paper announcement. First of all in the later case it is usually one set of

[33] http://www.nist.gov/tac/2013/KBP/index.html

[34] Such an event sometimes refers to as ***I**mplicit **R**elation **E**xtraction (IRE)*

Table 2.8. ACE Evaluation 2005 Event Extraction Task Summary

Event Type	Subtypes
Life	Be–Born, Marry, Divorce, Injure, Die
Movement	Transport
Transaction	Transfer–Ownership, Transfer–Money
Business	Start–Org, Merge–Org, Declare–Bankruptcy, End–Org
Conflict	Attack, Demonstrate
Personnel	Start–Position, End–Position, Nominate, Elect
Justice	Arrest–Jail, Release–Parole, Trial–Hearing, Charge–Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon

information par document with respect to the fact that news articles are not entirely focused on a single event. Moreover, news articles may not contain any formal structure that can be used to associate the information a system is trying to extract.

Summary

IE as a discipline comprise of its subtasks and these subtasks have a hierarchical dependency structure, i.e. for coreference resolution, NER is rather important if not absolutely necessary. Furthermore, generic IE practice adopts many of the IR and NLP methods and practices. In the next chapter we shall present the theory of measurement, the basis for the software size estimation theories and the state–of–the–art of automated software size estimation. We would specifically focus on the functional size measurement and we would like to argue that the conceptual model of functional size i.e. the data elements and their involvement in the transaction functions can be modelled using IE framework. Data elements are essentially files containing logical data grouping and often expressed in the requirement documents as named entities, i.e. field names and record names. The repeated use of these elements in a descriptive document is also a logical assumption, thus coreference resolution is not a far fetched idea in terms of modelling necessity. Finally transaction functions are by definition descriptions of the possible movements of the data elements, thus both relation extraction and event extraction seem logical possibility to model transaction functions. However we are addressing the automation problem of FPA from a higher level, i.e. identifying pages that can be selected for their FP contents and eventually reducing reading time for FP counters.

Like many complete IE systems we shall address the automation problem using IE and NLP methods that are most appropriate at this level. Our objective is to address the possible solutions for the problem from a level specific approach. Thus this chapter shall act as the groundwork for the farther advancement and possible direction of the research. We shall address this issue as a part of the future prospect of our research. We did not address either IE or NLP in details, primarily because of our limited use of these disciplines. We thus presented specific concepts and practices in brief and locally as we used them. We would present the possible use of custom designed IE subtasks such as the template development and *Template Element* extraction from a theoretical point of view not to stray too far from the core objectives of our research. These tasks will heavily depend on the concepts that shall be presented in the next chapter, especially the generic model of functional size estimation model.

Chapter 3

Estimating Software Size

Measurement, especially precise and accurate measurement is a necessary requirement for human activities and to a greater extent, in the scientific methods. For example, knowing the approximate volume of a room can be a useful for a financial query such as, ‘‘*what is the cost of heating the room*” but more precision for the same measurement is absolutely necessary for a scientific query such as, “*how many oxygen molecules are there in that room*”. Scientific study of *Software Development* is not an exception in this aspect, thus accurate measurement of software size is an intriguing problem for both researchers and software development companies. One can even argue that the ability to measure software size in advance (i.e. estimating the size before development) can be the most valuable resource for large software development companies in terms of controlling resources for a large project. The following sections will provide an overview of the general principals of measurement, available measurements for software, functional size measurement and the state–of–the–art of automatic software size measurement. For the clarity from now on software size measurement will be referred to as software size estimation or simply size estimation, considering the fact that unlike the concept of measurement of most physical object, software size is a conceptual estimation rather than a hard physically measurable quantity.

3.1 Measurement

Measurement is a key component in most if not all systems that govern our lives. For example, the ability to measure the passage of time allows us to wakeup on time (using an

alarm perhaps) and the ability to measure distance along with time kipping allow us to drive our vehicles safely (e.g. knowing the velocity at any point of time and thus we can maintain speed limit), and on the same note, the ability to measure numerous properties of nature is a prerequisite to have that engine running in the first place. Every aspect of our lives is influenced by the concept of measurement in one way or another. Measurement is not only for the professionals in the field of science and technology, everyone uses it everyday, e,g, price is just one measurement assigned to goods that is being sold which, also allows the shopkeeper to keep track of sales and more importantly state of ones business in terms of profit and loss. Looking closely at it one may argue that the concept of measurement allows us to understand our world, interact with it and improve our lives.

3.1.1 What Is Measurement?

The concept of measurement can be seen as a means of quantify or qualify attributes of things that allows us to compare and contrast by the means of some predefined rule. We compare prices of items in a store, contrast the size of cloths (e.g. size *Large*, *Medium*, or *Small*) etc. An informal definition of measurement can be,

> Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. – Fenton and Pfleeger (1997)

Measurements provide us information about *attributes* of *entities*. Entities can be a real world object (e.g. a table, a car, a human being etc.) or an event (e.g. a journey, coding phase of a software etc.). We identify entities by their characteristics that help us identify the similarity or difference with other entities. These characteristics can be called the attributes. such as length of a table, or the colour of a car, or duration of a journey. The conventional usage of an entity and its attributes are often interchangeable. *Temperature* for example, is an attribute of a physical body or a large geographical region at a specific moment in time, on the other hand *hot* or *cold* are qualitative values assigned to infer contrast about temperature. However, it is more often than not that we use phrases like "*It is rather hot today*", which actually implies that the temperature of the geographic region where the utterance occurred is higher than average human body temperature. These somewhat relaxed use of the terminologies does not effect our day to day activities, nevertheless, is an incorrect use of the notion. Especially from the scientific point of view the improper use of such notions are actually unacceptable.

Another way of looking at measurement is that, it is the process of assigning numbers or symbols to a real world entity that provide information, useful for comparison and/or contrast. However, this is a rather subjective notion of measurement by employing layers of abstraction that reflects our world view. These assignments of numbers and symbols are not random, we assign them to reflect some comparative and/or contrastive relationships as we perceive them. For example, we use measurements like *tall* or, *medium* or, *short* to express the height of a person. It allows us to make contrast among the height of different people. Lets say a person who has been described to be 7 feet tall, allows us to somewhat perceive that person's physical appearance by comparing his height with respect to our own, even without seeing him. Unfortunately, we may have a very similar if not the same perception about a person who has been described to be 7 feet 3 inches tall. Thus, measurement is far from an objective and clearly defined concept, so, the most basic measures such as length can easily be debated over and throughout our history it actually has been. The solution is to better understand the science of measurement, just as measurement itself is a reflection of our understanding of the attributes of an entity.

3.1.2 The Science of Measurement

> Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it.
>
> – H.James Harrington, in CIO – Sep 1999

Science as a discipline attempts to find ways to measure attributes that are significant to our needs and allow us to make the associated concepts more understandable and thus controllable. If we have a way to measure an attribute already, the objective of science is to improve the measurement in terms of quality, accuracy etc. Measurements have been an important focus in all fields of science and technology and over time we have learned to measure things that have been previously unmeasurable. For example, air quality, or human intelligence, or in our case the size of a software. Although these measurements are not as precise as we would like them to be. At this point, we would like to establish a clear distinction between measurement and calculation, *measurement* is a direct quantification of an attribute e.g. the length of a table or the weight of a box, whereas *calculation* is inferred from the measurement we can take, e.g. energy cost for a room is calculated from the volume and estimated cost of heating per volume etc. Furthermore, some complex calculations may

require some other calculation first, for example, calculating energy rating for a house or an apartment requires the calculation of the surface area, volume, energy loss etc. first.

We measure attributes of entities on a daily basis, we use tools and principals that we take for granted without any concern for the scientific principals behind these measurements. For example, we measure temperature and with passing time ever so precisely, but the principals were understood and refined slowly in many steps over time. As our understanding improves we have developed the frameworks to describe temperature and tools to perform the actual measurement. People had the notion of *hotter* or *colder* long time ago, long before our understanding of the concept of temperature. However, only in 1593, Galileo Galilei started the modern development process of measurement of temperature (Middleton, 1966) and it is only in 1854 the modern notion of *Absolute Scale* of temperature was developed. Thus measurement seems to evolve over time, with a strong co–relation to our understanding of the attribute of the entity it is associated with. Weight and distance are most probably the two basic measurements humans developed at a very early stage in our history. It started with the most obvious means, body parts and our natural surroundings; early Babylonian and Egyptian records indicate that length was first measured with the forearm, hand, or finger and time was measured by the periods of the sun, moon, and other observable heavenly bodies (National Institute of Standards and Technology, 1991). As our scientific understanding evolved, in particular, development of number systems and understanding of mathematics, the measurements become more complex and allow us to develop measurement units suited for various needs. Only in 1790, the French Academy of Sciences, on request of the National Assembly of France started developing the metric system as we know it today. They assign the name *meter* for decimal based length measurement unit. However, it evolved towards more universal standard for this measurement, since 1983, it has been defined as "the length of the path travelled by light in vacuum during a time interval of $1 \Big/ 299,792,458$ of a second". (MacKay and Oldford, 2000).

Formal measurement theory was developed as a discipline from the physical science to answer questions like, how much knowledge about an attribute is needed before measuring it or what meaningful statements can be made about an attribute and the entity through the measurement etc. We shall be discussing the representative theory of measurement in this section, i.e. choosing a set of rules to measure consistently and establishing a basis to interpret the data that is representative of our observations in the real world. The measurement data is expected to reflect our world view in a way that the manipulation of the data should be able to preserve the observed relationship among entities. We perceive real world by

comparison and contrast rather than assigning numbers to things, i.e. through *empirical relations* that allow us to compare and contrast. For example, *longer than* is an empirical relation, given that for the entity pair $\{x, y\}$ we can observe in real world either *x is longer than y* or *y is longer than x*. Although it qualifies our previous analysis that, a relation as such cannot clearly compare two entities that are very close in length. Furthermore, a relation does not need to be binary, it can be unary as in *x is long*, although, it is far less informative than the *longer than* relation. Thus we can think of the process of measurement as a mapping process between the empirically observable relations to formal mathematical notations e.g. assign a numerical value to the length of different entities. The mapping is done in a manner so that the set of entities mapped to a set of assigned values maintain the empirical relations existed before the mapping process. Thus, in the case of *x is longer than y*, *x* will be assigned a value that be greater than the value assigned to *y*. This behaviour is called the representation condition, formally it states that for the empirical relation *longer than* mapped into mathematical relation "$>$", the measurement mapping M must map entities to numbers in such a way that empirical relations are preserved by the numerical relation so that *x is longer than y* if and only if $M(x) > M(y)$. Now we can resonate with Fenton and Pfleeger (1997) to formally define measurement as,

> Measurement is a mapping from empirical world to the formal relational world. Consequently, a measure is a number or value assigned to an entity by this mapping in order to characterize an attribute. – Fenton and Pfleeger (1997)

However, sometimes it is difficult to have consent on empirical relations, especially if personal preference is a part of the empirical reasoning, e.g. ratings of wine. This is a part of human nature when it comes to establish clear cut empirical relation (*John Locke* in his *An Essay Concerning Human Understanding* emphasized this exact issue in 1690). Nevertheless, in terms of measurement theory, these types of subjective assessments are not necessarily measures rather they establish the basis for empirical reasoning and property characterization so that formal measurement can be possible in the future. In formal terms, real world is the *domain* and the mathematical world is the *range* in the mapping scenario , thus a mapping must define the domain, the range and the rules of the mapping. The process also involves developing a model to formulate measurements. *Models* are abstractions of reality, that allows us to strip details from an entity and assess it from the desired perspective. For example, to measure the height of a person, one must specify, if shoes are allowed for the measurement, or should we measure the height of the hair, thus we are defining the model

of a person rather then the person as the entity being measured. The model also need to supplement the mapping domain, i.e. how the model relates to its attributes.

In most of our earlier examples the measurements were *direct measurements*, i.e. when the measurement does not involve any other attribute or entity e.g. length, height etc. In contrast *indirect measurement* can only be performed in terms of other attributes e.g. speed of a moving object can be measures only in terms of the time and distance measure. Many of the obvious direct measurements are in fact indirect measurements, e.g. when we measure temperature using a thermometer, we basically measure the height of the bar of mercury in the thermometer and get the associate temperature. The model oriented measurement process helps us understand what happened in the past and what exists now, however, it also allows us to predict what might happen in the future, thus these models are also considered to be predictive models. In many occasions the model of empirical relations between entities can be used to establish the possible measure for a set of given parameter and can be very useful for our understanding. The distinction between measurement for assessment and prediction is not clear all the time, if we use a globe to measure the distance between two points, it is a model based assessment of the distance. Nevertheless, this indirect measure can be used to predict the cost for a future travel.

Measurement for prediction requires some mathematical model that bridges the prediction to attributes that can already measure. As in the previous example, sometimes we can use the assessment model somewhat directly for prediction, although, usually the prediction model also depends on other model parameters. Predictions are based on some other assessment and the accuracy of the assessment often dictates the outcome of the prediction. It is also rather common to use a probabilistic model with the assessments as the model parameters or prediction map's domain thus introduces risk or level of confidence in the prediction process. The downside of probabilistic model is that, often these modes are data driven and based on some underlying distribution (e.g. Gaussian, or Poisson etc.) in the data, thus if that distribution is altered during the usage of the model, the prediction will less likely be accurate.

3.1.3 Measurement Scales

The primary purpose of performing a mapping between empirical relations and numerical domain is to ease the manipulation of data in a manner that is well established (i.e. mathematical principals) and well understood and use the results to draw conclusions about

attributes in the empirical system. For example, we measure the air temperature for a day and conclude whether it is a hotter day than the day before, thus the numbers tell us about the characteristic of air maintaining the parallel with the empirical observation. However, there can be different possible valid mappings and they pose different levels of restrictions on the kind of analysis we can perform. These differences can be well understood through the notion of a *measurement scale*, then we can use scales to determine the appropriate analysis. Measurement scale thus can be the combination of the mapping and the empirical and numerical relation systems, although, when the relation system (i.e. the domain and the range) is obvious we can refer to the scale by the mapping alone. (Fenton and Pfleeger, 1997) argued that an established representation and scales should try to answer the following questions,

1. How we determine that one numerical system is preferable over another?
2. How we understand if a particular empirical relation system has a representation in a given numerical relation system?
3. What to do when we have several possible representation (i.e. many possible scales) in the same numerical relation system?

Relation systems need not be numerical, symbolic representation can be valuable as well. Thus, answering the first question, numerical representations (especially real numbers) are preferable primarily because of the ease in manipulation and our understanding of the manipulation techniques. The second question is rather theoretical and in this section we are more interested in the third question, i.e. determining the most suitable representation for measuring an attribute of interest. Number of relation in a system and the number of possible representations are inversely related and this notion can be understood through a formal characterization of the types of scales.

Moreover, one relation system is *richer* than the other if all the empirical relations in the second are contained within the first, and the richer the empirical relation system, the more restrictive the representation system will be. Length measurement is a perfect example, we can measure length in inch scale (British imperial system) and there are equally acceptable measure in feet, meters, miles etc. The mapping between acceptable relations are known as *admissible transformations* and the restrictive the set of admissible transformations, the more sophisticated the measurement scale is considered to be. In the case of length measurement, the set of admissible transformations are very limited, as a matter of fact there is only one possible transformation, $M' = aM$, where a is a constant thus, the length measurement is rather sophisticated measurement scale. We have listed the types of scales ordered by the level increased richness below,

1. Nominal Scale
2. Ordinal Scale
3. Interval Scale
4. Ratio Scale
5. Absolute Scale

The detail description of these scale types can be found in Appendix B along with the notion of meaningfullness associated with each of the scale types. Scale types are defined with respect to the set of admissible transformation, however, it is misleading to give an example of an attribute without specifying the empirical relation system. Table 3.1 presented the basis on which each of the the empirical relation systems is refined to be categorized to be an example of a scale type.

Table 3.1. Scales of Measurement Summary

Type	Admissible Transformation	Examples
Nominal	One–to–one mapping from M to M'	Entity classes
Ordinal	Monotonic increasing function i.e. $M(x) \geq M(y)$ implies $M'(x) \geq M'(y)$	Preference, air quality, intelligence tests
Interval	$M' = aM + b(a > 0)$	Relative time, temperature (Fahrenheit and Celsius)
Ratio	$M' = aM(a > 0)$	Time interval, length, temperature (Kelvin)
Absolute	$M' = M$	Counting entities

3.1.4 Meaningful Measurements

Establishing scale types allows us to determine the meaningfulness of different statement that can be made about the measurement of an attribute of an entity. It has already been established that computing ratio with nominal, ordinal or interval scales, although the use of real numbers often seem tempting to manipulate them in a familiar manner i.e. averaging, addition or even statistical analysis. The knowledge and understanding of the scale type allow us to perform the proper analysis and deduce meaningful statements about the attribute of the entity that has been measured. It is important to understand that the meaningfulness of a statement involving measurement is quite distinct from the notion of the statements truth, For example, the statement "*My father is 500 years old*" is nevertheless a meaningful

statement about the measurement of age however from a factual point of view it is clearly untrue. Thus formally the meaningfulness can be described as, a statement is meaningful if the truth value is invariant of transformations of allowable scales. the following examples will try to shade some light on the meaningfulness,

"x" is twice as tall as "y"

This statement implies that the measures are at least on a ratio scale, since scaler multiplication is required for an admissible transformation. This statement is meaningful because all possible measures of height (e.g. inch, feet, centimetre etc.) have no effect on the truthfulness of the statement. Formally, if M amd M' are two different measures of length, both $M(x) = 2 \times M(y)$ and $M'(x) = 2 \times M'(y)$ are either true or false and this consistency is due to the relationship $M = aM'$ for $a > 0$.

Temperature of Paris is twice as that of Oslo today

This statement also implies ratio scale but not meaningful in Celsius or Fahrenheit scale, however, it will make clear sense in Kelvin scale. Thus the meaningfulness of this statement depends on the scale being used and how much information we originally intended to convey.

The temperature difference between Paris and Oslo today is twice as that of yesterday

This statement implies that the difference between temperature for two different days are meaningful and a part of the conditions for interval scale or any other scale of higher information content. Let us consider, yesterday's temperature of Paris and Oslo was respectively 15°C and 10°C so the difference is 5°C. In accordance to the statement let us assume today's temperature was 20°C and 5°C respectively i.e. the difference is now 10°C, just twice as yesterday. It is meaningful for any interval scale for temperature e.g. Fahrenheit scale. However, the statement is not true for the same temperature in Kelvin scale, although the information content is higher.

Meaning lies within the interpretation of the total scenario i.e. measurement scale, its level of information content and nevertheless intention for measurement. Meaningfulness is often clear when the measure is familiar, however, if we have to define a new measurement it is not always clear. For example, carbon–dating technique to measure the age of fossils may not seem an obvious way to do so, or practical, or even easy, but the measures are certainly valid and meaningful. Meaningfulness thus, should be viewed as one of the attribute of a measure.

There is also the perception of measurement that affects the meaningfulness namely subjective and objective measurement. It is commonly accepted and understandable, the need

of objective measurement, however, subjective measurement is sometimes informative especially when objective measurement is not a practical possibility. For example, if one wants to establish a measurement for the quality of wine, the objective measures based on perhaps multiple attributes of a wine can understandably equal if not less informative than the subjective measure of quality ranking made by wine experts. It is a common practice to ask the experts to comment or assign ranks to different properties of the wine, although subjective, they are capable of providing valuable information about a wine. As long as we treat a subjective measure with the consideration of its subjective nature, many complex measurement can be made with a certain degree of confidence with them.

3.1.5 Indirect and Extended Number Measurement

In many situations it is not possible to measure an attribute directly (software size estimation can be a good example), thus many of our very common measurements are either indirect or force us to use extended number based measurement. Let us consider an optimization problem (Fenton and Pfleeger, 1997) presented in Table 3.2,

Table 3.2. Transportation Quality Assessment

Transportation	Journey Time (Hours)	Cost Per KM (Euros)
Car	3	1.5
Train	5	2.0
Aeroplane	3.5	3.5
Executive Coach	7	4.0

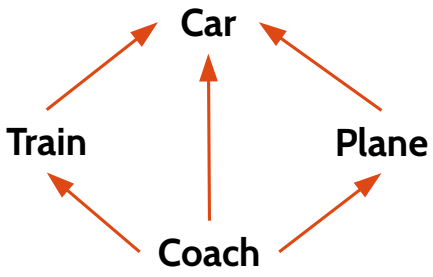
Let us try to assess the quality of transportation from the data to travel from one city to another. In the given scenario quality has two sub–attributes, *journey time* and *cost per KM*. The question we are trying to answer is, given two transportations A and B, how to rank A is of higher quality than B. We can establish a simple rule to make the statement "*A is better than B*" when,

$$journey\_time(A) < journey\_time(B) \textbf{ AND } cost\_per\_km(A) < cost\_per\_km(B)$$

We can depict the quality relations between different transportations using the aforementioned rule as presented in Figure 3.1. Each outgoing arrow from entity B to entity A represents

the higher quality of A over B. For example, in Figure 3.1 all the other transportation is better that *coach* and *car* is better that all the other transportation. However, *train* and *plane* are not comparable with this rule. We need more information, under the condition some subjective measure, such as, whether we prefer speed over cost or the other way round to make conclusive analysis.

Fig. 3.1. The Quality relation Between Different Transportation



We can easily prove that we cannot convert this measurement into a single–valued real number measure. If we want to map the quality of *plane* and *train* into real number through the maps $m(plane)$ and $m(train)$ respectively, it must satisfy one of the followings,

1. $m(plane) < m(train)$
2. $m(plane) > m(train)$
3. $m(plane) = m(train)$

The first statement proves that cost is the decisive factor, and time is for statement two. However there is no way to establish statement three. The reason we cannot find a measure because for a real incompatibility we have partial order thus we cannot map the relation to a set of real numbers $\Re$. What we need instead is a mapping into a pair of real numbers, i.e. in a set of $\Re \times \Re$. We can define the mapping, $m(transport) = (journy\_time, cost\_per\_km)$ e.g. $m(car) = (3, 1.5)$ or $m(train) = (5, 2)$. The relation now can be defined as follows and this numerical relation preserves the empirical relation and it is a partial ordered value in $\Re \times \Re$, since it contains incomparable pairs.

$$(x,y) \textbf{ superior to } (x',y') \textbf{ if } x < x' \textbf{ and } y < y'$$

On the other hand, indirect measurements establish a single valued outcome using multiple directly measurable components i.e. sub–attributes, which can eventually be used for other indirect measurements. Indirect measurement must also abide by the same basic principles of measurement theory that we use for direct measurements. Thus, indirect measurements

are bound by the restrictions of scale types i.e the admissible transformations. For indirect measurement let us define admissible transformation as a rescaling. Formally, for the indirect measure M we measure n sub–attributes $M_1, M_2, \ldots, M_n$ i.e. $M = f(M_1, M_2, \ldots, M_n)$ for some function f. We can call M' a *rescaling* of M if there are rescaling of $M'_1, M'_2, \ldots, M'_n$ for $M_1, M_2, \ldots, M_n$ respectively, so that $M' = f(M'_1, M'_2, \ldots, M'_n)$. For example,

The measure for *density d* is an indirect measure of *mass m* and *volume V*, the relation is expressed in the form,

$$d = m/V$$

So, every rescaling of d is in the form $d' = \alpha d$ for $\alpha > 0$. We have to demonstrate two things: function of this form is rescaling, and all the rescaling are of this form. For the first part, we have to find rescaling m' and V' for m and V, respectively such that $\alpha d = m'/V'$. Since, both m and V are in ratio scale, so αm and V are acceptable rescaling of m and V respectively since,

$$\alpha d = \alpha\left(\frac{m}{V}\right) = \frac{\alpha m}{V}$$

Thus, we have rescaling. For the second point, since m and V are ratio scale measures, every rescaling of m and V must be in the form of $\alpha_1 m$ for some α_1 and $\alpha_2 V$ for some α_2. Therefore, the rescaling of d has the form,

$$\frac{\alpha_1 m}{\alpha_2 V} = \frac{\alpha_1}{\alpha_2}\left(\frac{m}{V}\right) = \frac{\alpha_1}{\alpha_2} d = \alpha d \ldots (\text{where } \alpha = \frac{\alpha_1}{\alpha_2})$$

The scale type for indirect measure is defined exactly the same way for direct scale. In case of the mixed scale type present in the sub–attributes' scale types, the indirect measure will generally not be stronger than the weakest scale in the mix, e.g. if a mix contains ratio, interval and nominal type, the resultant scale will be at best nominal. In the next section we focus on different measurement possibilities for software size that is intrinsically related to the notions of measurement presented above.

3.2 Software Size Estimation

Estimating the software size before and during the development process is becoming a fundamental requirement in the ever–growing modern software development industry. It has been used as a valuable tool to keep track of development cost, effort distribution and productivity, especially for large scale development projects. Furthermore, measuring the size of a software at different stages, starting from the earliest possible specification can

provides a comparative benchmark of the projects progress and evolution. Thus reasoning dictates that the earlier a stage the size measure is taken the more valuable it can be to the overall development planning. We would like to establish the scope of this work that it is pertaining software size estimation and not cost or effort estimation. The cost or effort calculation is the subject to a whole separate brunch of research on their own rights and use the estimated software size as primary input (independent of the origin of the size in terms of the type of metric being used to obtain it).

We would like to further extend that the scope of the research that it is our objective to investigate metrics, regarding the size of a software, often classified to be a subset of *Product Metrics*. Product metrics relate directly to the result of a software development process. Important features of the product that are often measured include (although not exclusively): size, quality, user requirements, product growth, and user comfort (Bundschuh and Dekkers, 2008, pp. 208). Many of the software measurement related literature considered *McCabe's Cyclomatic Complexity* (McCabe, 1976) to be coherent to the topic of size measurement. However, in our humble opinion although complexity measure is considered to be a product metric, it is rather loosely related to estimating the size. The metric is better known for complexity analysis of a system and further detail can be found in the work of Madi et al. (2013).

Among the different metrics presented in the following subsections, the more implicative metrics are the source line and functional size based metrics. In relation to the previous theoretical discussion about the different measurement scales, both these metrics were developed with the goal of conveying maximum amount of information, i.e. absolute scale. At the size estimation level at least, they are often used to justify statistical meaningfulness of the measures. Thus, it is logical to assume that there exit a linear relation between a source based estimation (S) and functional estimation (F) in the form $S = aF$, Albrecht and Gaffney (1983) and later Gencel and Demirors (2008) addressed this issue in details. In the following subsections, different types of software size metrics will be presented in details.

3.2.1 Source Line of Code

***S**ource **L**ine **o**f **C**ode (SLOC)* most probably is the oldest and rather obvious means to measure the size of a software since, all software eventually end up being lines of code. During the early days of computer programming, while using punch cards to provide the codes to a computer system, it was the intuitive way to measure the size of a software. In the early

days it was also the most reasonable since, most of the effort and cost used to be spent on programming since the dominant programming language was *Assembly*. However, once the higher level languages (e.g. FORTRAN, ALGOL 58, COBOL, PL/1 etc. during the 60's) started to emerge and becoming the dominant development environment, the primary effort started to shift towards specification and algorithm design. Although, it seems obvious how the size estimation with SLOC works, it actually is more complicated. There are many ways to measure the SLOC, e.g. one can measure all the lines with blanks, comments etc. or one may count only the significant lines i.e. no comment or blank lines. The later is often referred to as ***E**ffective **S**ource **L**ine **o**f **C**ode (ESLOC)*. Regardless of the inherent weaknesses (see (Jones, 1978)), it is widely used in many cost estimation models e.g. Boehm et al. (2000) adapted this framework to use in the popular COCOMO model. The primary reason for this is that SLOC has been shown to have strong correlation with the software cost; thus, they are relevant inputs for software size estimation models Boehm (1981); Boehm et al. (2000).

The most significant shortcoming of SLOC being lack of standardized counting guideline. Over the year researchers attempted to establish guidelines for the ESLOC, since raw SLOC is well established and less useful. (Boehm et al., 2000) in his COCOMO book presented a definition of ***D**elivered **S**ource **I**nstructions (DSI)* or ***D**elivered **S**ource **L**ines **o**f **C**ode (DSLOC)* to be used as the size parameter for COCOMO model. The IEEE standard for *Software Productivity Metrics*[1] also provided definitions and attributes for SLOC sizing metrics. (Park, 1992) along with the *Size Subgroup* of the *Software Metrics Definition Working Group* and the *Software Process Measurement Project Team* of the ***S**oftware **E**ngineering **I**nstitute (SEI)* at *Carnegie Mellon University* extended the SLOC metrics into a counting framework known as the SEI framework. It contains a set of counting definitions and a check–list to use the guideline so that there could be a well defined, consistent and repeatable SLOC measurement. Although, due to the focus on what to count rather than when and how many, it caused multiple interpretation and ambiguity in the counting tools development from this framework.

Nguyen et al. (2007) presented Table 3.3 in argument of the necessity of proper definition and guideline for counting SLOC. Jones (2004) listed that there were at least 75 commercial software cost estimating tools exist on the market at the time of his research and at least 20 SLOC counting application. As demonstrated in Table 3.3, they tend to produce quite different logical SLOC counts especially if the project is long. Nguyen et al. (2007) also made the observation that the ratio also varies due to the programming practices and programming

[1] 1045–1992: IEEE Standard for Software Productivity Metrics

Table 3.3. SLOC Counts of Large Files From Open Source Products

Products	Physical	CodeCount™[a]		RSM[b]		LocMetrics	
		Logical	Ratio	Logical	Ratio	Logical	Ratio
OpenWBEM	14,000	7,100	1.97	4,700	2.98	6,600	2.12
FlightGear	14,000	10,800	1.30	7,600	1.84	9,900	1.41
wxWidgets	50,300	30,700	1.64	21,300	2.36	27,300	1.84

[a]Center for Systems and Software Engineering, University of Southern California
[b]M Squared Technologies$_{LLC}$

style. However, SLOC is not without its merits, First and foremost as mentioned earlier, since it can be very organized given a standard counting method, SLOC based metrics can be easily and consistently measured using software tools. The basic model definition presented in Putnam (1978a) is as follows,

Based on the Putnam (1978b) model, the SLOC count is made for each part of a software system after breaking it down to smaller and more manageable sections or pieces. Putnam (1978a) presented that there are three distinct estimate for each part, a (Smallest possible SLOC), m (Most likely SLOC), and b (Largest possible SLOC). The estimated SLOC count for each part E_i is calculated using the following formula,

$$E_i = \frac{a+4m+b}{6}$$

The expected SLOC for the whole system with n parts thus be,

$$E = \sum_{i=1}^{n} E_i$$

The estimate of the *Standard Deviation* of each of the estimates E_i can be obtained by getting the range in which 99% of the estimated values are likely to occur, i.e.

$$SD_i = \frac{|b-a|}{6}$$

Once SLOC is estimated, it can be used in a cost model to estimate the possible cost of the software. (Putnam, 1978a) presented that along with the SLOC count in thousand SLOC's or

KLOC, three other variables are needed, α (the marginal cost per KLOC), β (an exponent of the KLOC), and γ (the additional fixed cost of the project). Thus, the estimated cost is calculated by the following formula,

$$CostEsitmate = \alpha \times KLOC^{\beta} + \gamma$$

This is just the fundamental overview of the SLOC based size and cost estimation. Each model uses its own rational and formulas to estimate these and other possible measurements. Nevertheless, the downside to the metrics based on SLOC for large projects is that it is only viable once the coding is completed, although, for cost estimation, effort estimation and other project management tasks, estimating the size at earlier stages of development is more valuable. Moreover, programming style and programming language may not have much effect on the raw SLOC count but as shown in Table 3.3, it heavily affects the logical SLOC count which is the basis for most SLOC based estimation metrics. SLOC is still effectively being used in many of the popular software measurement metrics and the tools based on them.

3.2.2 The Theory of Software Science

Maurice Howard Halstead in his 1977 monograph *Elements of Software Science* (Halstead, 1977) attempted to describe the empirical science of software development. According to his theory a computer program is considered in *Software Science* to be a series of tokens which can be classified as either *operators* or *operands* and he attempted to establish a metric to quantify complexity directly from them. The metric is considers to be one of the oldest metric that measure software size from the source code i.e. complexity was defined as a function of the *operators* and *operands*. The metric is based on four absolute scale measures,

1. n_1 : number of unique operators
2. n_2 : number of unique operands
3. N_1 : number of total operators
4. N_2 : number of total operands

Using these basic elements, the metric is defined to calculate a set of measures relevant to software size and complexity estimation process. Resonating with Shen et al. (1983), we

believe that Halstead (although he had never stated explicitly) conceptualized his mode of software as the collection of a programmer's thought process to manipulate an unique set of operators and operands. These basic assumptions later led Halstead to hypothesize a complete metric. In Appendix C we presented the metric in details along with the criticism faced by it from the scientific community.

3.2.3 ABC Metric

Jerry Fitzpatrick (1997) introduces this metric in an attempt to overcome the shortcomings of SLOC and other similar metrics. He theorized that high level languages (e.g. C, PASCAL etc.) have three fundamental operations, storage or **A**ssignments for explicit transfer of data to a storage location i.e. variables, **B**ranching for forwarding program branches out of current scope and **C**onditions for a logical test. These **ABC** values are represented as ordered triplet of integers i.e. $ABC = \langle a, b, c \rangle$, where a, b and c are assignment, branching and condition counts respectively for a given code segment. He provided the following formula to calculate the single valued representation of the software size,

$$|ABC| = \sqrt{a^2 + b^2 + c^2}$$

The generic condition and reflections of the resulting value has been explained by Fitzpatrick (1997) as, it is the best when for a single module the value is less than or equal to 10 and upto 20 it is acceptable. If however, the value is between 21 and 60 some re–factoring is required and it become unacceptable when it is greater than 61. The metric is very easy to calculate in a automatic manner and since it uses the logical structure of a program, it is virtually independent of the programming style of a programmer. Moreover, it can easily be applied at different segmentation level of a program i.e. it can be applied for a subroutine, or a package, or a class definition etc.

On the downside, the metric reflects the working size of a program rather than the actual length of a program i.e. equal size programs in terms of ABC metric may vary in actual size depending on the expressiveness of the programming language used to code. Moreover, it may give a *zero* value for a program not performing any of the A, B or C operations regardless of the fact that it is performing something e.g. if a piece of code is just printing some messages on the screen. ABC metric is not very widely adopted for size measurement in the software development community.The other significant attempt to overcome the weaknesses

of the SLOC based metrics is the functional size measurement and we shall present it in the following subsection.

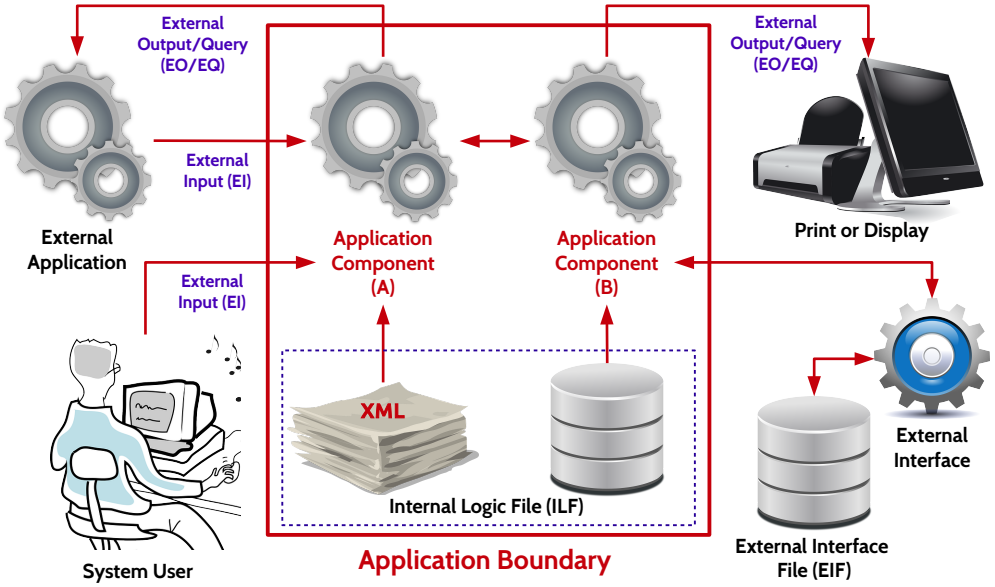
3.3 Functional Size Estimation

Function **P**oint **A**nalysis (FPA) was proposed by Allan Albrecht of IBM in 1979 (Albrecht, 1979) for functional size estimation as an alternative to the SLOC based methods. This method measures the size of a software by quantifying the functionality of a software that are meaningful to the software users. Originally, investigated as a means to measure software development productivity, it eventually became an effective means for software size estimator. The most lucrative aspect is the possibility of applying this method across a wide range of development environments and at any stage of the life-cycle of a development project, from early requirements definition to full operational use. However, even today, identification of **F**unction **P**oints (FP) from early stage specification documents, is still almost entirely a manual effort, slow, and time consuming, especially for large projects. Moreover, large organizations either have to maintain experienced personnel dedicated for the function point counting or acquire the expensive services of the organizations dedicated to the task. The counting accuracy also depends on the size of *project base* maintained by the counting body (Kemerer, 1993). In summary, the manual counting process suffers significantly from speed and consistency due to the lack of automation.

The metric was originally developed as an alternative to SLOC as the entry point to measure productivity at the later stage of the software development life–cycle. Albrecht (1979) argued that it can also be a powerful tool to estimate the size and thus cost of a software project at an early stage (as early as user requirement analysis stage) of the development life–cycle. Furthermore, due to the objective nature of the functionality identification approach, anyone with the working understanding of the software functionality and development, can perform the estimation task without the explicit knowledge of the design and development consideration for a specific project (Kemerer, 1987). Another significant advantage is the programming language and programming style independent nature of the method. For example, to estimate the size accurately using SLOC the programming language is an important consideration since if a program is written in a high level language e.g. *Java* will undoubtedly differ significantly in size if instead is written *Assembly* language (Kemerer, 1987). A Proper understanding can however, only be possible by exploring the process of **F**unction **P**oint **A**nalysis (FPA) i.e. the raw FP counting, adjustments performed, how the

final FP count is calculated and most importantly how the final FP count is interpreted. We would like to make the distinction that FP count is a relative measure of the software size based on many factors such as, programming practice of an organization or type of project (e.g. new development or maintenance) etc.

Fig. 3.2. Functional Perspective of an Application



FPA is in a way intuitive (see Figure 3.2) since we humans are accustomed to solve problems by breaking them down to smaller, meaningful and useful pieces and we then classify them into categories that group those pieces using some features of those pieces. It is a familiar process and used in every day situations, e.g. librarians organize books based on different features as long as they are meaningful, books here represent the minimum meaningful units and the categories they are put into are the classes. The process allows some feature(s) to be used to establish meaningfulness in the classification process and give rise to rules that to be followed to obtain consistent results. To parallel it to the concept of FPA, we would like to argue that software by definition is intangible thus SLOC or other metric like that can only capture the possible indirect tangibility of the design and implementation decision choices made during the software development but not the software itself. Whereas, FPs are the intangible equivalent to unit parts of a piece of tangible object (the unit being the minimal measurable unit for a specific scale[2]) i.e. FPs represent the minimum meaningful

[2]we can measure length with any ratio scale that we can reasonably construct. Thus, any measurement can only be meaningful for the purpose of compare and contrast, upon the agreement on the scale itself. In case of intangible concepts the options are often limited, e.g. it might turn into a purely philosophical discussion if we want to list all possible scales to measure the size of a intangible piece of software.

units of a software from the point of view that eventually each software is a collection of functionality that it is capable of performing. Thus, Albrecht (1979) presented the process of classifying each functionality of a software into a specific group on the basis of features of the functionality, effectively using common sense to perform a rather complicated task.

3.3.1 Albrecht's Productivity Measurement

Before becoming a dominating software size metric, FPA was Albrecht's (Albrecht, 1979) solution to measure productivity in IBM®'s ***D**ata **P**rocessing (DP)* services organization. SLOC was the preferred metric at that time for different estimation metrics and researchers were aware of the obvious shortcomings of SLOC. IBM was using multiple programming languages (e.g. PL/1, DMS/VS, COBOL etc.) for its development projects already and a programming language independent and accurate productivity estimation metric was a desperate necessity for a organization of that size[3]. In this classic article (Albrecht, 1979), Albrecht described the state of operation of the DP services organization having around 450 people dealing with 150 – 170 projects at any given time. He also mentioned his observation that design phase was about 20% of the total effort whereas 80% of the effort was for implementation. Thus, one must measure the process including the design phase. Table 3.4 summarize the dataset used for the research,

Table 3.4. Dataset (Albrecht, 1979)

Language	**Projects**
COBOL	16
PL/1	4
DMS/VS	2
Total	**22**

All these projects were complete projects through all the phases, from the requirement analysis to the final system test and demonstration. They were also been completed under DP project's management with consistent task definition and management procedure. The work hours invested on these projects were all accounted for and the functional factors were known. He summarized that "the application function was consistently proportional to a

[3]in 1979 during Albrecht's work. IBM had a net income of 3.01 billions with 337,119 employees and 696,918 stockholders (see "IBM HIGHLIGHTS, 1970 – 1984" for the history and significant events in IBM between 1970 & 1984).

weight count of the number of external user inputs, outputs, inquiries and master file". The notable fact is there were originally four functional elements defined by Albrecht (1979) rather than the modern five class FPA (see § 3.3.2). He also explained the method to count each class of functionality delivered by a development project. These factors, as he argued, are the perceivable manifestation of any piece of software. The counting process for the classes were explained as follows,

◦ **Input Count:** *Input* provides business function communication from the user to the computer system, i.e. data forms, terminal screen, keyed transactions etc. He proposed not to count an input more than once and count all unique inputs. The uniqueness of an input implies that inputs for every different processing logic are unique. Furthermore, if an input does not provide any functionality and used rather for transitional purpose, it should not be counted. Finally, regardless of the input like nature, inquiries and files should be distinguished and counted separately.

◦ **Output Count:** *Output* provides business function communication from the computer system to the user, i.e. printed reports, terminal outputs etc. Like inputs, he argued for the counting of all unique external outputs, i.e. either an output having a different format than others or requires unique processing logic to be generated. He however, proposed not to count simple error messages or the acknowledgement of an entry i.e. outputs that does not require any distinct processing i.e. enquires.

◦ **Inquiry Count:** *Inquiries* were defined to be the input & output pairs where the inputs are for control purpose only and and the generated outputs do not require any explicit processing. Each enquiry with uniquely formatted or uniquely processed output from a file search should be counted.

◦ **File Count:** Each machine readable logical file or logical grouping of data from the user's perspective that is used or maintained by the system, e.g. files stored in any media should be counted. Major data groups within a database should be counted, however only logical files are to be counted but not the physical manifestation for the files i.e. if there is an indexing file for faster access it should not be counted since it is a part of the same logical file. Furthermore, all machine readable interfaces to other systems should be counted as files.

Once the initial count is completed, each count value is weighted on the basis of the significance of a function type to a user and over debate and trial. The multipliers presented by Albrecht (1979) have been summarized in Table 3.5. The results are then further adjusted for other factors, e.g. complex input, output and files were adjusted by adding 5% extra or

Table 3.5. Primary Adjustment Multipliers (Albrecht, 1979)

Function Type	Multiplier
Input	4
Output	5
Inquiry	4
Files	10

complex internal processing being adjusted by adding another 5% etc. However, maximum allowed adjustment was upto $\pm 25\%$ thus, producing a dimensionless value representing the effective relative measure for the functional value to be delivered to the customers or users.

The evaluation was designed to establish the relation between development cost in terms of work–hours used for each function point for a project with some other variable. He effectively showed that between 1974 and 1980, the trend (linear least square fit trend line) is a negative slope thus, indicated increase in productivity (about 200%). He also showed that project size is proportional to the par FP work–hour requirement thus claimed that project size is a component of the FP counting process. The final take can be the following,

1. Disciplined software development effectively influence the development process thus, the co–relation with FP count.
2. Functional measurement can be done cross–languages with strong co–relation to the actual implementation effort.
3. Inclusion of project effort at every level of the development process is absolutely necessary to acquire realistic functional measurement.
4. Functional measurement allows comparing projects without the dependencies such as programming language and programming style thus provides a means to perform objective fact analysis.

This work however presents the findings of the analysis of 22 projects and further research and standardization was suggested for more generic and independent analysis of project of different size nature thus pushes the eventual standardization of the method.

3.3.2 IFPUG Function Point Analysis

The formal restructuring of what eventually became the modern form of FP counting was presented in Albrecht and Gaffney (1983) and the next year Albrecht compiled the first

counting guideline[4]. The US based, user govern, non-profit ***I**nternational **F**unction **P**oint **U**sers **G**roup (IFPUG)* was formed shortly after that in 1986[5] to standardize and maintain the method itself, explore its application and documenting counting practices. IFPUG method has two components, first component deals with the measurement of the functional size based on the original work of Albrecht (1979) and the other component measures the contribution to overall size of 14 technical and quality factors. There has been several variations of the original idea, nevertheless, IFPUG ***F**unctional **S**ize **M**easurement (FSM)* is arguably the most popular method. IFPUG periodically released the ***C**ounting **P**ractice **M**anual (CPM)*, the guideline for all IFPUG counting professionals and the latest version is IFPUG CPM 4.3.1 and it was released in January, 2010. The fundamental layout of the counting process remained somewhat unaltered throughout different versions, however, these alternations will also be addressed in this section.

FPA is predominantly an intuitive perception of a software system thus captures very fundamental components of a piece of software. FP counting begins with the identification of the conceptual logical boundary i.e. *application boundary* that represents the actual entity to be measured. Furthermore, there are *five* major components defined for a software system and counting professionals attempt to count each of these five types of function points. There are *two* file types (functions correspond to logical blocks of data) and *three* transaction types (functions correspond to data movement) and they are as follows,

1. ***I**nternal **L**ogic **F**ile (ILF)*
2. ***E**xternal **I**nterface **F**ile (EIF)*
3. ***E**xternal **I**nputs (EI)*
4. ***E**xternal **O**utputs (EO)*
5. ***E**xternal **I**nquiry (EQ)*

Furthermore, during the counting process a complexity measure is also associated with each function point type derived from the three following parameters,

1. ***D**ata **E**lement **T**ype (DET)*
2. ***R**ecord **E**lement **T**ype (RET)*
3. ***F**ile **T**ype **R**eferenced (FTR)*

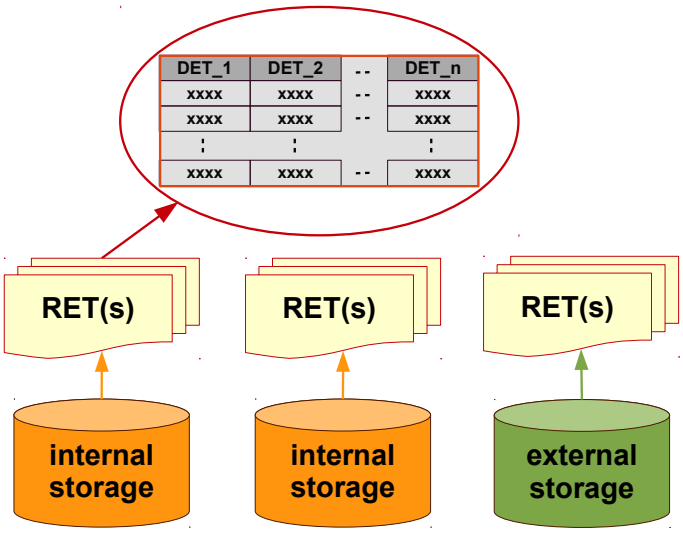
Each **DET** is a unique user–recognized non–recursive data field i.e. information that is dynamic but associated with the single data field. In case of a recursive DTE only the first

[4]IBM CIS & A Guideline 313, AD/M Productivity Measurement and Estimate Validation, dated November 1, 1984 (Smith, 1997)

[5]http://www.ifpug.org/

occurrence is considered. DETs can be viewed as a column of a table (see Figure 3.3), thus represents the minimum class representation of data. The counting guideline suggests caution in counting each user–recognized unique DET just once, preferably the first occurrence. Screen elements that are static like system time or page counters are not counted as DETs, however, any single–action link to submit the content of a form is counted as a DET. Each data input field, error messages, and calculated values are also counted as a DET for a input transaction (EI and EQ). Each Data Field on a report, calculated values, error messages, and column headings that are read from a file are considered DET for a output transaction (EO and EQ).

Fig. 3.3. Function Point: Complexity Parameters



Each **RET** is a user–recognized logical sub–group of DETs that may exist in a file e.g. tables in a database (see Figure 3.3). The counting guideline suggests that, if a set of tables are closely related e.g. composite relation, they should be counted as a single RET. However, other tables in a relational database being related by a common key should be counted as individual RETs. If a parent child relation exist, each sub–group in a parent group is counted to be a different RET.

FTR is the files (either ILF or EIF) that have been referenced by a transaction function. It is the logical grouping of RETs exist in a local storage or accessible via remote interface in the form of a file. For each reference to a ILF or EIF one FTR is counted.

ILF is an expansion on the original file count (Albrecht, 1979) characterized by all logically related data that is kept and maintained within the application boundary. Table 3.6 shows how the complexity of each recognized file is estimated,

Table 3.6. Individual File Complexity

RET(s)	DET(s)		
	1–19	20–50	50+
1	Low	Low	Avg
2–5	Low	Avg	High
5+	Avg	High	High

EIF, the other expansion, in contrast are user definable group of logically related data that are completely outside of the application boundary. EIF's are used by the measured application but purely for reference purposes i.e. these data are maintained by some other application. Thus, EIF's are ILF's for some other applications. The complexity is estimated using Table 3.6 just as for ILFs. Once all the files are identified the individual complexity is then used to identify the weight multiplier for each file type to be multiplied by the raw frequency of each type–complexity pair (see Table 3.7).

Table 3.7. File Weight Multiplier

Complexity	Values	
	ILF	EIF
Low	7	5
Avg	10	7
High	15	10

EI is counted when a functionality implies to an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application and it may also be used to maintain ILFs. The data can be either control information or business information. If the data is control information it does not have to update an internal logical file. The complexity of each EI is then determined by a predefined complexity metric (e.g. see Table 3.8).

EO type is counted when a functionality implies to an elementary process in which derived data passes across the boundary from inside to outside. Moreover, an EO can update an ILF and uses ILFs and/or EIFs to generate outputs for the user or to send to other applications. The process involved is expected to explicitly derive the output rather than just extract the information from some file. The complexity of each EI is then determined by a predefined complexity metric shared by both EI and EQ (e.g. see Table 3.9).

Table 3.8. Individual EI Complexity

FTR(s)	DET(s)		
	1–4	5–15	15+
0–1	Low	Low	Avg
2	Low	Avg	High
2+	Avg	High	High

Table 3.9. Individual EO & EQ Complexity

FTR(s)	DET(s)		
	1–5	6–19	19+
0–1	Low	Low	Avg
2–3	Low	Avg	High
3+	Avg	High	High

EQ is counted as a pair of input and output that perform the function of extracting some data from the storage (both ILF and EIF) and send it outside of the application boundary. The input part does not update any ILFs and the output does not contain any derived data. Once all the transactions are identified, the individual complexity is then used to identify the weight multiplier for each transaction type to be multiplied by the raw frequency of each type–complexity pair (e.g. see Table 3.10).

Table 3.10. Transaction Function Weight Multiplier

Complexity	Values		
	EI	EO	EQ
Low	3	3	4
Avg	4	4	5
High	6	6	7

Once all the counts are accounted for and each multiplied by the corresponding weight multiplier, summing them up gives a dimension less integer known as the ***Unadjusted Function Point (UFP)***. This concludes the first part of the FSM–IFPUG counting process. In the IFPUG–CPM, the next step has been presented as determining the ***Value Adjustment Factor (VAF)***. However, it has also being presented as an optional part of the FP counting process in the most recent IFPUG–CPM. We described the idea in details in Appendix D.

At this point, FP counting may seem very structured and well defined, but the truth cannot be far enough. Identifying ***B**asic **F**unctional **C**omponent (BFCs)* can be tricky even when following a well defined and extensive guide line such as IFPUG–CPM. The process tends to be very subjective at both individual counters and organization level. Due to the highly subjective nature, the identification is often imperfect and varies upto 10.78% (Kemerer, 1993) from one counting professional to another when counting FPs for the same application. Other researches showed that this variation exists among the counters from the same organization following the same operational procedure (Kitchenham, 1997) and the variation can be upto 30% and it is even higher between counters from different organizations (Low and Jeffery, 1990). The data provided by IFPUG showed that certified counters vary upto 12% for the same application (Buglione, 2008). Furthermore, a significant amount of effort is invested to analyse several heterogeneous requirement documents in order to identify BFCs that are often not documented properly.Thus, in case of a discrepancy, it is hard to trace back to the BFCs presence in the original document set (del Bianco et al., 2008) making any modification nevertheless a painstakingly difficult process. The complexity of the process give rise to various different practices of the original idea that has been briefly presented in Appendix E. We however, used the IFPUG methodology as the standard methodology for this research. The next section is dedicated to the stat–of–art of the automation in FPA and our observation regarding the presented works.

3.4 Automatic Function Point Analysis

Defining the automation of *Function Point Analysis* poses an unique problem since the development of the FP counting process has been targeted to restrict subjectivity rather than explore objectivity in the description of a software. Furthermore, the documents traditionally used for the task are different levels of software requirement and description documents written in NL by people with different levels of technical competence, who are working in organizations with a whole range of development practices. Thus, a single functionality of the a software can be documented in many different ways and can also be interpreted in different manner. FPA provides a layer of abstraction to the possible variations and allows a human FP counter to identify a specific type of functionality in all the different representations. An automatic FP identifier will then face the problem of the limitation of understanding the language of the document, which often can be tricky even for humans. Very little research has been done in total automation of FPA, especially in FP recognition. One of the key reason for

that is the stringent requirements (although not unfair) of the existing certification standards and nevertheless the complex nature of the task. This section will provide an overview of the problem, available standards and the little research we manage to find that has been done.

3.4.1 IFPUG Software Tool Certification

IFPUG Software Tool Certification is somewhat the only available standard that is acceptable in the FPA community (al least in the IFPUG community) and frequently used to classify a software that performs in part or the whole process of the FPA. This standard nevertheless provides a definition of the automation standards and it is called the IFPUG *Software Tool Certification* types. Currently there are *three* types of certification available,

◦ ***Type 1 – Function Point Data Collection/Calculation:*** The software provides Function Point data collection and calculation functionality, where the user performs the Function Point count manually and the software acts as a repository of the data and performs the appropriate FP calculations.
◦ ***Type 2 – Expert System That Aids Counting of Function Points:*** The software provides Function Point data collection and calculation functionality, where the user and the system/-software determine the Function Point count interactively. The user answers the questions presented by the system/software and the system/software makes decisions about the count, records it and performs the appropriate calculations.
◦ ***Type 3 – Automatic Counting of Function Points:*** The software carries out an automatic Function Point count of an application using multiple sources of information such as the application software, database management system and stored descriptions from software design and development tools. The user may enter some data interactively, but the involvement during the count is minimal. Furthermore, the software and its associated documentation must conform to the Counting Practices Manual. Software Type 3 instructions and criteria are currently under review by the IFPUG Board of Directors.

Furthermore, consulting the IFPUG Software Certification page[6] one can notice that there are only *three* software ever to receive a type one certification, only one software (which also have a type one certification) have type two certification and no software ever received a type three certification. Table 3.11 summarizes the status of software certification by IFPUG during the writing of this document (May 3rd 2015).

[6] http://www.ifpug.org/certification/software-certification/

Table 3.11. IFPUG Software Certification Status

Type	Software	Developer	Certification Date
Type 1	Function Point Workbench	Charismatek Software Metrics	July 1998
	PQMPlus	Q/P Management Group	November 2001
	Software Metrics Manager	Softmet.com	April 2004
Type 2	PQMPlus	Q/P Management Group	November 2001

3.4.2 Automation of Functional Measurement

The research for automating the process of FP counting (both partial and complete) is quite old. Initial studies focused on either the automation possibilities or defining the levels of possible automation. MacDonell (1994) studied nine functional size measurement methods and estimation models and reported that all of the FSM methods were assessed against six criteria including the automation criteria. The automation criteria rated, on average, one of the lowest for all the FSM methods. He concluded that automation of FSM methods required further research. Mendes (1997) actually suggested 10 levels of automation instead of 3 currently in practice by the IFPUG. He studied 8 ***C**omputer **A**ided **S**oftware **E**ngineering (CASE)* systems, for all the vendors claimed to support FP as a part of their system (Jones, 1996). Mendes (1997) found that only one vendor had a feature to calculate automatic FP counts from source code (COBOL). The survey revealed that no vendor claimed that they could automatically count the steps of the FPA without external intervention.

April et al. (1997) summarize the aforementioned patterns along with the observation made at that time that, there are several work published describing some automatic FP implementation details. Some of them were from the IFPUG conferences, e.g. works of Brown (1990), Mazzucco (1990, 1992) etc. and some from industry publications, e.g. Banker et al. (1994), Sample and Hill (1993) etc. They (April et al., 1997) also reported that at that time there was no known literature that describes reliability, validity or precision of the results of those implementations. They thus, concluded that invalidated automation tools with undocumented results implies only the unreliability of those tools, at least slowed down the possible deployment and usage of such tools in the real world. They (April et al., 1997) however, presented a method to formalize and evaluate IFPUG counting rules to perform automatic FP count from source code. The next subsection will focus on other works on FP counting from source code (many of them used the work of April et al. (1997) as the launching platform).

3.4.3 FP from Source Code

There has been some effort to automate the FP counting from the source code, especially for enhancement projects. One of the earlier literature was the work of Ho and Abran (1999), proposing a framework that can be used to build a model of automatic FP counting from the COBOL source code, in compliance with the IFPUG–CPM. They presented a code slicing technique to develop a tool to perform the task. Klusener (2003) on the other hand presented a method to model the FP counting rules instantiated for COBOL and ***J**ob **C**ontrol **L**anguage (JCL)* with the suggestion that using the data flow analysis to improve the counting of transaction functions. Once again it was suggested by the authors that the method needed to be validated by running through real project environment.

Ellafi and Meli (2006) presented an improvement over Ho and Abran (1999) method in terms of the architecture driven FP extraction and pairing it with the ***E**arly & **Q**uick (E&Q)* FP counting method. This hybrid method, relying heavily on code analysis, was proposed to be logic driven and the integration of the E&Q FP counting was presented to be an improvement over existing method by accelerating the counting process, and ensures consistency and significance of the functional measurements. However, the authors presented it to be a semi–automatic framework that empowers the FP measurer by giving control of the counting process via a semi–automatic mode that leveraged the information retrieved from the source code analysis. Although, possible supporting tools were proposed, there was neither any mention of an actual implementation nor any experimental results were presented.

Some methods were developed for other languages as well, for example, Sneed (2000) presented a critical analysis of the FPA principals, especially, IFPUG–FPA (even compared the practice to "religion") and a methodology specifically designed to count FP from source code written in C++ or Java following object oriented design. However, the details at which the method has been presented, no evaluation has been presented thus left us at best sceptical about the method. Later, Kusumoto et al. (2002) also presented a method to extract FP count from Java code. There method was designed to agree with the IFPUG counting rules and mapped the object method interaction hierarchy to each of the function point type. Although a case study was presented in the literature, the method was reported to have never been tested.

Kusumoto et al. (2008) later presented a method to identify FPs for web applications focusing on the screen transitions and database access using ***S**tructured **Q**uery **L**anguage (SQL)*. They propose to use screen transitions and SQL access to identify transition func-

tions whereas, proposing to use SQL calls to form "*SQL tree*" and map database tables to extract data functions. Although the authors claimed to developed a tool for identifying FPs from application codes writen in Java and using *Apache Struts*[7], no experimental data has been presented. Moreover, they presented their doubt about the methods applicability for applications developed outside of the *Struts* framework.

3.4.4 FP from Other Environments

There has been some research on estimating the FPs from other environments of the software development life–cycle. Banker et al. (1994) for example, presented a methodology to estimate the functional size from a integrated CASE based development environment. It is imperative to understand that regardless of the objective of achieving automation for the FP counting process, all these methods relies on the use of a specific practice (often accompanies the requirement of using specific development environment and tools) for the whole development process. The methodology adopted by Banker et al. (1994) used the features from the CASE environment to achieve the automation. They used a repository object–base with indexing integrated with the CASE environment tool that represents the organization of a software into objects and thus facilitates the FP counting process with clearly defined object interaction without analysing any code at all. They also presented a software reuse analyser measure that can complement the final FP count, however, they did not present any large scale evaluation of the system only to left the readers with the optimistic prediction of a possible 80% cost cut through the use of the tool and development environment.

Uemura et al. (1999) presented a somewhat generic method, by proposing the use of UML design specification. They developed a tool for the task that used the UML specification version 1.1, IFPUG–CPM version 4.0 and Rational Rose[8] version 4.0. They primarily used the class diagrams and the sequence diagrams with FP determination rules for the task. They presented a method to determine the data functions by a 3 step strategy: selection of candidates, determining function types (ILF or EIF) and finally determining the complexity. On the other hand, transaction functions were counted using a similar strategy with the exception of of the use of 5 predefined patterns for determining the function types. Although they presented a case study of 3 simple systems the results were inconclusive at best. The

[7] https://struts.apache.org/
[8] http://www-03.ibm.com/software/products/en/ratirosefami

authors themselves proposed more larger scale experimentation to determine the validity of the tool and the method.

Lamma et al. (2004) presented a rather similar approach to the problem and attempted partial automation of the FP counting process using ***E**ntity **R**ealtion – **D**ata **F**low **D**iagram (ER–DFD)*. They proposed to automatically identify data and transaction function and the complexity to determined the unadjusted FP count where the FP count type and application boundary is already known. They used a modified form of the IFPUG counting rules (they claimed to have ignored the ambiguities in the IFPUG counting rules) to be expressed in terms of the properties of ER–DFD and formulate the rules. They then used the rules translated in Prolog for their tool. From the authors claims it is unclear how the system performs in real world situation regardless of their rather bold claim of the tool (called FUN) being a type 3 IFPUG tool (it has never been certified to the best of our knowledge). They presented a case study of using the tool in 5 partially overlapping projects (projects from a single portfolio). However the only project (they claimed it to be a large project: 257 FPs counted by a professional) the tool has been used to be validated by human counter reported to have performed exceptionally well. They however, express the necessity to validate the tool with more projects and proper evaluation.

A progression of the work by Lamma et al. (2004) in many ways presented by Fraternali et al. (2006) using the WebML development environment (Ceri et al., 2000) and for web application size measurement. WebML is a *Model Driven Development (MDD)* environment and defines a development project as a conceptual model using many of the frameworks available for example, ***E**ntity **R**elation (ER)* model is commonly used for data representation. The described method is fundamentally re–visits the method presented by Lamma et al. (2004) while using different set of tools and a redesigned rule–set. The presentation of the experimental results were not conclusive since only 4 application (144, 300, 513 and 680 FP respectively counted by a professional) was tested upon and showed near impeccable performance. They also claimed that their method counts more accurately than the human counters especially when a duplicate pruning algorithm was used. The systems results varied a maximum of 11.11% with that of the human counters.

A more recent work by Batista et al. (2011) presented a system called ***R**equirement **M**odel **F**unction **P**oint (ReMoFP)* that uses a process designated "*Problem Modelling*", a UML based requirement modelling from the users point of view. This particular perspective coincide with the original philosophy of the functional measurement more closely. Their system used user specification defined by ***O**bject **C**onstraint **L**anguage (OCL)* for constraint definition and

UML extensively. They evaluated the system by closely monitoring the development of one project (2245 FPs) and monitoring the change in FP counting productivity. They reported 107% productivity improvement over standard IFPUG count by the professionals. However, they did not present any results concerning the counting accuracy. The system heavily relies on the UML definition of a project and concerned mostly with the FP identification (just as the other systems). They emphasized on larger scale experimentation and evaluation as well.

Choi et al. (2006, 2012) proposed a goal–scenario based approach to the automation of FP counting from requirement documents in natural language. The presented method however, was conceptual and never been experimented other than of course the case study that was presented and no tools were developed to automate the process. The goal–scenario modelling form the requirement was based on the work of Kim et al. (2006) and the resultant goal–scenario tree was used along with the FP extraction rules for acquire the final UFP count. They evaluated the approach with 4 rather generic and small projects (the projects had between 53 and 78 FPs) and reported to have worked very well with an average accuracy of 94% with respect to that of the professional counters. The important aspect of this work is the use of NLP and IE techniques such as, the use of syntactic pattern based templates to model the goal–scenario tree of a project. Although the authors expressed the possibility of the development of tools to automate the process, further literature did not show up during our research.

Summary

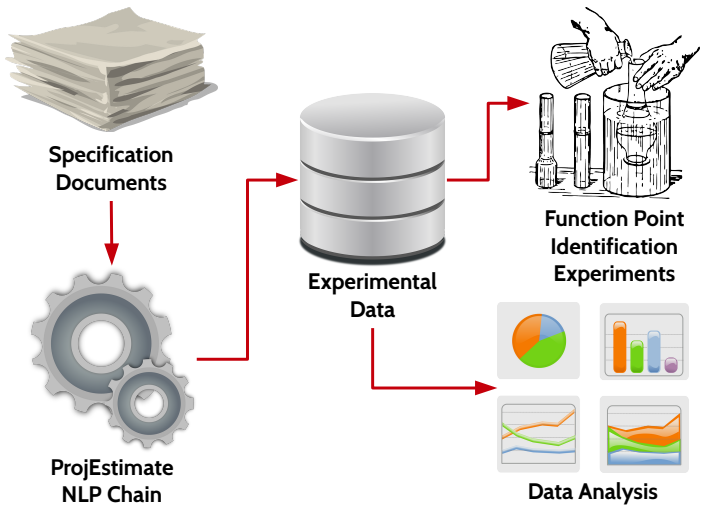
We have presented our understanding of the problem domain and the state–of–the–are for FPA and automated FPA in this chapter. We shall proceed to present our contribution which has been spread over the next two chapters. We have produced the only data resource that can be used for automated FPA using both linguistic and data engineering approach within the scope of the *Projestimate* project. We have seen the general concept of measurement been applied to establish the foundation for different estimation metrics. Functional estimation is not only the practical approach in our case but also the only metric that we can use to calculate FP at the early stages of the development life–cycle. Since significant amount of literature is unavailable addressing the problem of identification of FPs from specification documents, we are assuming our contribution to be novel and significant in terms of continuing research.

Chapter 4

Text Extraction and Analysis

The two primary contributions of this research are proposing suitable methodologies for the task, e.g. identifying higher level textual groups such as pages, paragraphs etc. that contain FP descriptions and developing a processing chain to extract, normalize and annotate (morphology & syntax) a corpus automatically for ***F**unction **P**oint (FP)* identification research. The general overview of our contributions is illustrated in Figure 4.1. Since automation is

Fig. 4.1. General Overview of Our Contributions



quite uncommon in the current practice of FP counting, identifying pages with FP descriptions for example, will facilitate the task (making it easier) in terms of reducing reading effort for the FP counters. It will also improve consistency in redundant count of the same specification by the same counter (or different counters) since each time the counter will read the same set of pages. In this chapter we shall describe the methodology and the tools developed to extract

the corpus from the source documents, the design decisions made during the development of the chain and a general presentation of the generated data. The source documents are early stage software specifications, primarily user requirements and functional specifications. We would like to emphasize the fact that, the corpus we have produced is the basis for this research and the future research on FP identification within the scope of the *ProjEstimate* project.

4.1 The Source Data

The data used in this research is collected under the *ProjEstimate* project (§ 1.4), provided by two of the project partners *Banque de France (BdF)* and *PSA Peugeot Citroën Group (PSA)*. The raw data is comprised of complete or partial project documentation of different types e.g. user requirements, functional specification, batch processing description, database design (***E**ntity **R**elation & **D**ata **F**low **D**iagrams (ER–DFD)*) and ***U**nified **M**odeling **L**anguage (UML)* based design specification etc. The digital contents consist of documents in *Microsoft Word* format (both old **.doc** and the newer **.docx** format), *Rich Text Format* (**.rtf**), *Microsoft Excel* Format (**.xls**) and *Microsoft PowerPoint* Format (**.ppt**). The scope of this research is to process ***N**atural **L**anguage (NL)* text that contain FP descriptions in a single document exclusively, thus, the preliminary selection of the data was based on the following considerations,

- The document types having less text concerning FP descriptions e.g. ***.ppt*** files or files presented in a structured format e.g. ***.xls*** files were discarded.
- Documents heavy on graphics and diagrams e.g. conceptual specifications and UML diagrams, were discarded as well.
- In case of the presence of multiple documents for a single project, each document was processed as individual data units.
- Database specifications containing relational database specifications were discarded since these documents are often graphic "*heavy*" and they are primarily useful if used in a multi–document setting for establishing document interrelations.

We did not make any distinction between development and enhancement projects because we found that in large organizations such as *BdF* and *PSA*, even many new projects often rely on existing software infrastructure and old projects. Although, this distinction between development and enhancement projects however has significant implication in the traditional FP counting process. Our justification for the choice is that we are interested in the identification of FP descriptions rather than the counting specifics. Regardless of the counting type,

the existence of the FPs is identical for both types of projects. The data that has been used in the research, being from two different organizations also ensures data diversity and thus, demands additional robustness for all the tools being used in the analysis stage. A general overview of the datasets will be presented in the following subsections. Unfortunately, due to to confidentiality restrictions we are unable to mention the specifics or the names of the projects and we shall address them by the assigned designation for this research.

4.1.1 Data Provided by BdF

BdF provided 4 project specification documentation for 2 enhancement and 2 development projects. We also have the detail FP count worksheets for these projects and we found that there is *one* large project with 4000+ FPs, *two* medium projects having FP count between $700-800$ FPs and *one* smaller project with 300+ FPs. The FP counting for all of these projects were performed by the same counting professionals. Table 4.1 summarizes the general overview of the data.

Table 4.1. BdF Provided Data Description

File Designation	Specification Type	Project Type	Pages	FP Count
001.bdf	Functional Specification	Enhancement	158	4,291
002.bdf	Functional Specification	Development	31	319
003.bdf	Functional Specification	Enhancement	39	739
004.bdf	Functional Specification	Development	65	785

4.1.2 Data Provided by PSA

PSA provided 13 project specification documents, but unlike the *BdF* dataset we have no information regarding the FP counts for these projects. Among the 13 projects we have used only 9 project documentations and a total of 14 files. We have found multiple files are useful in some projects and many of them are related specification from some other old project. Many files were rejected primarily because they are either too short in terms of pages (anything below 15 pages was rejected) or they contain mostly graphics and tables. One can make the observation that *PSA* dataset is a rather mixed bunch whose page size ranges from as small as 19 to over 300 pages and comprises both functional specification

and user requirements. We have found that the presence of graphics and tabular elements are the primary contributor to the size of the larger files. In Table 4.2 we present the general overview of the data.

Table 4.2. PSA Provided Data Description

File Designation	Specification Type	Pages
005.psa	Functional Specification	195
006.psa	Functional Specification	25
007.psa	Functional Specification	312
008.psa	Functional Specification	107
009.psa	User Requirement	34
010.psa	Functional Specification	79
011.psa	User Requirement	33
012.psa	User Requirement	19
013.psa	Functional Specification	69
014.psa	Functional Specification	94
015.psa	Functional Specification	55
016.psa	Functional Specification	57
017.psa	User Requirement	241
018.psa	Functional Specification	105

4.1.3 Annotated Data

We also have *one* document, designated `000.ref`, where the texts that contains FP descriptions have been annotated by *four* different professional FP counters. It is a *functional specification* for a project from *PSA* which is 37 pages long and contains a moderate amount of graphics and tabular contents. On first inspection we found that it contains many tables which include descriptive texts and all the annotators highlighted at least some part of these texts to have FP descriptions. The annotation was performed using the guidelines developed during the early stages of the *ProjEstimate* project (§ 1.4) in collaboration with the FP counting professionals from the partner organizations. Annotations were made by highlighting different types of relevant pieces of texts with different colours. A detail of the annotation guideline can be found in Appendix A (§ A.1).

Among the *four* annotators, *one* is from *ACAPI*, a member organizations of the *ProjEstimate* project,(designated `Acapi_JNV`), *two* from *BDF* (designated `BdF_BD` and `BdF_MG`) and *one* from *PSA* (designated `PSA`). The primary statistics of the annotation has been summarized in Table 4.3. Although not conclusive, it is hard to miss the resemblance in the pattern of FP

Table 4.3. Primary Annotation Statistics: 000.ref

Annotator	Annotation Count				
	FUN_GRP	APP	FUN	DATA_GRP	Total
Acapi_JNV	2	4	14	8	28
BdF_BD	0	0	69	9	78
BdF_MG	0	0	49	50	99
PSA	8	2	32	22	64

annotation among the counter from the same organization (`BDF_DB` and `BDF_MG`). Another observation is the large amount of inconsistency among the counters (even when they are from the same organization) while comparing the types of annotation found in the same block of text. Once we had the basic linguistic analysis done we performed inter–annotator agreement analysis at page (*Pg*), sentence (*Sent*) and token (*Tok*) level with *three* rigidity levels (`L0`, `L1`, and `L2`) of agreement. The complete analysis has been summarized below.

Table 4.4. Inter–Annotator Agreement: 000.ref

Annotator		Annotator								
		Acapi_JNV			BdF_BD			BdF_MG		
	Level	Pg	Sent	Tok	Pg	Sent	Tok	Pg	Sent	Tok
BdF_BD	**L0**	0.69	0.07	0.05	-	-	-	-	-	-
	L1	0.53	0.04	0.02	-	-	-	-	-	-
	L2	0.23	0.02	0.02	-	-	-	-	-	-
BdF_MG	**L0**	0.86	0.04	0.03	0.69	0.66	0.71	-	-	-
	L1	0.73	0.04	0.03	0.69	0.63	0.67	-	-	-
	L2	0.34	0.04	0.03	0.53	0.52	0.63	-	-	-
PSA	**L0**	0.69	0.14	0.08	1.0	0.52	0.49	0.69	0.47	0.57
	L1	0.69	0.07	0.03	1.0	0.35	0.31	0.69	0.35	0.38
	L2	-0.07	0.07	0.03	0.03	0.33	0.30	-0.07	0.34	0.37

The rigidity levels for the agreement count is defined on the basis of matched elements and the number of annotation type associated with a single block (i.e. page, sentence or token) that is being matched. Although, an agreement does not require identical annotations in an

annotation pair, a matching annotation within a block is the basis for a potential match and each match is then filtered according to the rigidity level. The different rigidity levels are defined as follows,

- **Level 0 (L0):** It is the most flexible level and any match regardless of type or number of types is considered an agreement. In a unit element i.e. page, sentence or token, if an annotation exists for both annotators within a block regardless if there is a partial intersection of types or disjoint types, a match is counted. We would like to call it agreement due to vicinity.
- **Level 1 (L1):** It is a moderate flexibility level accepting matches of the same type for at least one pair of annotations within the block. Since one element may contain more that one type of annotation, this level allows agreement on minimum matching.
- **Level 2 (L2):** It is the most strict level. If an annotation is found in a block by both annotators, the algorithm compares all the types of annotation made by both and only if there is a complete agreement (i.e. both annotators annotated exactly the same types of FP) then a positive match is counted.

We have used *Cohen's Kappa Coefficient* (κ)(Cohen, 1960) to measure the inter–annotator agreement. We overlay the rigidity levels during the calculation as a filter. Kappa is a statistical measure commonly used to find the agreement between two annotators, each classifying N items into C mutually exclusive categories. Our case had a little difference with the original definition since one entity can be annotated for more that one class, thus the rigidity transformation is used to converts the annotations into a binary classification case, i.e. each entity can either be classified as "*annotated*" or "*not–annotated*" which in turn depends on the rigidity level. The general formula to calculate κ is,

$$\kappa = \frac{P(a) - P(e)}{1 - P(e)}$$

where, $P(a)$ is the relative observed agreement among annotators and $P(e)$ is the hypothetical probability of agreement by sheer chance (if the annotators assign classes randomly). If there is a complete agreement $\kappa = 1$ and if there is no agreement among the annotators other than chance, $\kappa = 0$. κ can also be negative if the relative agreement is less than the agreement by random chance.

An important observation (although not conclusive) can be made from Table 4.4 that an overall high agreement has been observed among the annotators from the same organization (e.g. the annotators from *BdF*, marked in "*blue*") and the poorest agreement has been observed among `Acapi_JNV` and the other annotators. Using Table A.1 we can argue that the agreements correlate with the number of total annotation that exist in the first place for the

annotators. Another interesting observation is that the agreement between PSA and BdF_BD seem to be perfect at page level for rigidity level L0 and L1 but it is very poor at L2. This implies that even though the annotators made annotations on the same pages and agreed on at least one type of annotation, they disagreed on the finer details, i.e. they did not agree on the existence of all the types of annotation with each other. Furthermore, the agreement between PSA and BdF_MG at page level for rigidity level L2 is negative i.e. their agreement, on FP types at least, is even less than chance could have produced. On the other hand the annotator pair from *BdF* has been demonstrated strong agreement even at token level and at every rigidity level. We would like to acknowledge that the data were not collected in a controlled environment, thus, we cannot objectively comment on the level of collusion during the actual annotation process, especially among the annotators from the same organization, that may have compromised the independence of the annotation among annotators. However, we are assigning relative confidence and importance in the data considering this data represents the whole evaluation scope for our research and as of today, an unique resource of its kind to the best of our knowledge.

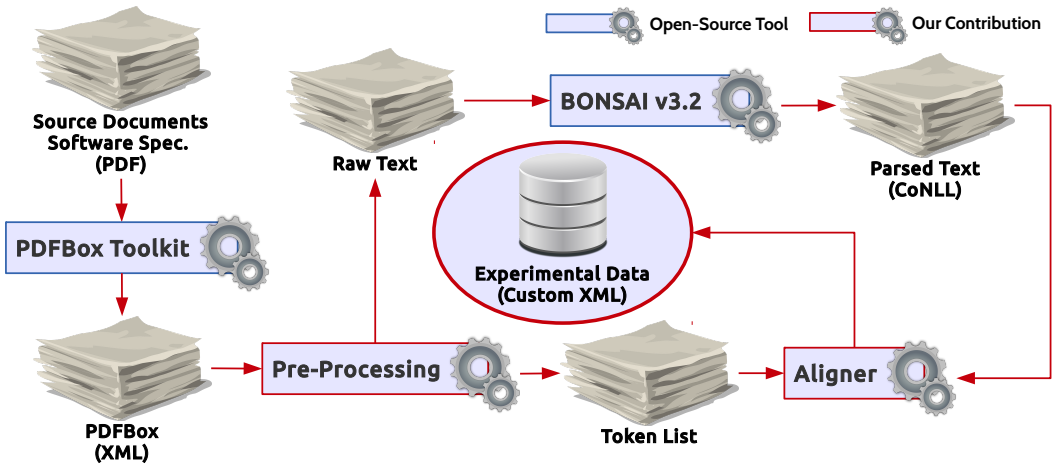
4.2 Corpus Development from Source PDF

Text extraction from different digital content is an active field of research but our focus was to extract text and significant formatting information from *PDF* files rather than advanced technological aspects of the process. *PDF* is a common document format for electronic documents primarily because it preserves the original look and feel, i.e. it maintains low level objects such as group of characters, lines, curves and images and associated style attributes such as font, color, stroke, fill, and shapes, etc. However, PDF documents were originally designed for the final presentation of a document, thus, usually do not contain the higher level logical structures such as words, text lines, paragraph, figure illustration etc. There has been significant amount of research on content understanding and extraction, block segmentation and identifying logical structures in PDF files (Chao and Fan, 2004). However, for this research we have used already established, freely available text extraction tools.

The first components of corpus development is extracting the texts from documents in digital format (our raw data was in the popular **.doc**, or **.rtf** format) and the second component is the preprocessing (normalize) the extracted text and performing basic linguistic processing. The resultant data is machine readable text in a custom *XML* format defined for the convenience of information representation for the research. For the text extraction, we have used the

***P**ortable **D**ocument **F**ormat (PDF)* as a standard input due to the simplicity of converting the raw data (in different formats) into PDF format and the popularity of the format as the de–facto standard for the exchange of print–oriented documents (Hassan and Baumgartner, 2005). The target language for this research is *French* and all our source data is software specifications written in the target language. Furthermore, all the tools that has been used in this research were specific for the target language (unless a tool is language independent). Figure 4.2 shall provide the general outline of the corpus generation process. A language

Fig. 4.2. General Outline for The Corpus Generation Process



independent text extraction framework has been used for the initial extraction from PDF documents (*Apache PDFBox*). The extracted text is quite noisy since all the text elements, regardless of their relevance, are extracted thus, some pre–processing is required. The text is then run through a parsing chain to acquire basic linguistic knowledge. The parsing chain also provides multiple levels of textual segmentations e.g. tokens, sentences etc. from the extracted text. The final output then can be formatted to produce the ***F**unction **P**oint **A**nalysis (FPA)* corpus in a custom XML format, with all the generated information. In this section we shall provide an overview of corpus development and general statistics at the extraction stage.

4.2.1 The PDF Format

A *PDF* file is a binary file format made up of a sequence of bytes. These bytes, grouped into tokens, make up the basic objects upon which higher level objects and structures are built (see ISO-32000–7.3). The organization of these objects, how to read and write them is defined in the file structure of the PDF and in addition, a file can be encrypted to protect

the document's content (see ISO-32000–7.5). Within the file structure, basic objects are used to create a document structure through building higher level objects such as pages, bookmarks, annotations (see ISO-32000 7.7). There are two widely used low level PDF object models: ***S**tructured **D**ocument **F**ormat (SDF)* and ***C**arousel **O**bject **S**ystem (COS)*[1]. *SDF* is the acronym used in `PDFNet-SDK`[2], whereas *COS* is used in `Adobe Acrobat SDK`.

The *SDF/COS* object system provides the low–level object types and file structures used in PDF documents i.e. PDF documents at the lowest level are graphs of *SDF/COS* objects. These objects can represent document components such as bookmarks, pages, fonts, and annotations. However, using such low level data structure is often tedious and error prone since the programmer must remember all of the names of the parameters and helper methods are not usually available. So, many PDF manipulation libraries (e.g. *PDFBox*) provide higher level models (e.g. the ***P**DF **D**ocument (PD)* model for *PDFBox*) that allows different *SDF/COS* objects (e.g. page, font, image etc.) to have predefined set of attributes that is available in an object dictionary. Moreover, for each class, strongly typed methods are available to access the attributes, thus, reducing the programming complexity. For example, each class of the *PD* model in the *PDFBox* library often represents a single *COS* object providing methods to read and write both content and attributes (meta–data).

4.2.2 Extraction Tools

We have explored the possibility of using *two* content extraction tools, both are open–source libraries from *The Apache Software Foundation*, namely *Apache Tika* and *Apache PDFBox*. There are many text extraction tools for the PDF format available (e.g. *PDFlib* Text Extraction Toolkit, *PDFNet SDK*, *LEADTOOLS PDF SDK* etc.) that can perform the task of extraction with varying levels of details and accuracy. However, we were interested in a open–source library that will allow us to extract unlimited amount of text with some generic context information e.g. font formatting, paragraph boundaries etc. Each of these two tools were experimented with our processing chain but eventually, we decided to incorporate *Apache PDFBox* in the processing chain for some significant advantages in extracting formatting information, which is specially useful during the preprocessing stage. Moreover, *Apache Tika* is fundamentally a common detection and parsing wrapper for many formats that uses *Apache PDFBox* for parsing and extraction of PDF files.

[1] *Carousel* was the codename for *Acrobat 1.0*
[2] https://www.pdftron.com/

Apache Tika is a content analysis toolkit that allows analysis and content extraction from 1000+ different file formats and outputs in several formats including, text, *HTML*, *XML* etc. It is a *Java* library that also have a stand alone command line tool that can be used for text extraction.There are *two* primary interfaces for this toolkit: *Detector*, the basis for most of the content type detection and *Parser*, accommodating different file format information and parsing subroutines to parse different types of file. *Tika* is widely used in search engines, especially in the crawler modules to extract text and meta–data from digital contents. *Apache Tika*, however, provides a common interface to parse all the different file formats. We used the command line wrapper for the extraction experiments.

Apache PDFBox library is also an open–source *Java* library for working with PDF documents. *Apache PDFBox* is published under the Apache License v2.0 and has several features in addition to extracting (e.g. Splitting, Printing etc.). According to *OpenHUB*[3], the development summary of *Apache PDFBox* (as of May, 2015) was:

- The project has had 3,998 commits by 17 contributors representing 120,668 lines of code.
- It is written in Java with an average number of source code comments (68,418 lines).
- It took an estimated 31 man years of effort (counted using *COCOMO* model) starting with its first commit in February, 2008.

Along with the Java library *Apache PDFBox* also comes with a series of command line utilities that can be used for the aforementioned tasks. *Apache PDFBox* command line tool for text extraction outputs in either text or HTML format.

4.2.3 Text Extraction

The first experiments on text extraction was conducted using the *Apache Tika* command line tool to extract text in the `XML` format. The tool is also capable of producing `HTML` content which can preserve more format information, especially table structures for some tables. On the other hand we used the *Apache PDFBox* command line tool to produce the `HTML` content from the *PDF* documents. Since the tool cannot produce *XML* output directly, we then used the "*xmllint*" parser to produce the `XML` output. Both types of *Apache Tika* produced outputs (`XML` & `HTML`) contains a lot more meta–information than the *Apache PDFBox*. Furthermore, both tools produce the physical line boundaries rather than logical boundaries such as sentences or paragraphs. However, except for some minor formatting details the text content is outputted quite in the same manner. However, *Apache PDFBox* is more

[3]https://www.openhub.net/p/pdfbox/

specific, lighter and less resource intensive among the *two* tools. The general statistics for the extracted files is listed in Table 4.5. The green row represents the largest file and the light red row represents the smallest file (for details of the files see § 4.1). It is to be noted that the

Table 4.5. Extracted File Summary

File	Lines	Words	Characters
001.bdf.html	14,918	65,968	481,651
002.bdf.html	1,972	8,734	67,150
003.bdf.html	3,321	12,311	94,750
004.bdf.html	5,831	25,255	198,192
005.psa.html	19,963	65,876	496,029
006.psa.html	2,129	6,054	48,539
007.psa.html	30,647	118,047	854,149
008.psa.html	7,786	27,584	213,332
009.psa.html	2,595	9,872	78,012
010.psa.html	5,740	20,942	161,702
011.psa.html	2,468	11,890	86,428
012.psa.html	1,518	5,650	45,921
013.psa.html	6,166	20,673	158,731
014.psa.html	11,402	55,855	401,857
015.psa.html	4,216	15,731	127,208
016.psa.html	3,356	12,825	103,898
017.psa.html	14,209	85,622	589,872
018.psa.html	25,130	106,511	757,736
Total	**163,367**	**675,400**	**4,965,157**

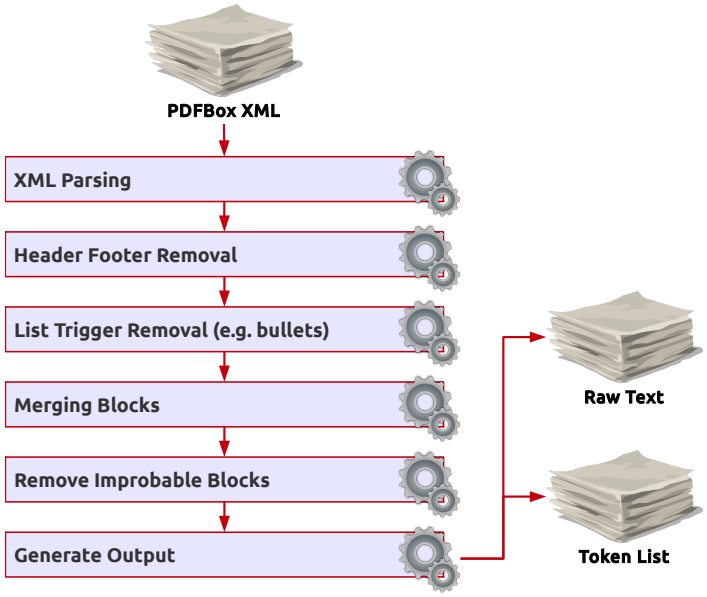
extracted files contain large amount of *HTML* tags produced by the extraction utility, thus, the size is not representative of the actual text content of each file. The extraction utility also removes all graphic content thus, many large files eventually yields disproportional amount of text.

4.3 Preprocessing

Once the text is extracted from the source documents, the first tool we developed is the preprocessing utility. The extracted text was found to be far from clean since all the formatting information extracted by *Apache PDFBox* is pertaining to the physical structure of the

document rather than the linguistic frames (i.e. paragraph, sentence, words etc.). So, we developed a preprocessing tool, written in *Python*[4]. The tool has been tested on *Python* 2.7 on *Ubuntu* 14.04 LTS (`Trusty Tahr`) and developed as an executable script expected to be executed in a command line environment. Most of the components are heavily parametrized and can be used to control the behaviour of the program and to accommodate parsing and annotation mapping (if any) in the later stage. *Two* types of output (i.e. raw text for parsing and token map with annotation) were produced by the pre–processing tool. The process flow is depicted in Figure 4.3. The text preprocessing stage is a single script containing all

Fig. 4.3. Overview of The Preprocessing Tool



the modules. The output generation module can be modified for any future requirement to produce other types of outputs. Each Module has explicit dependency on the previous module except for the *XML* parser of course, and they are executed in the exact sequence presented in Figure 4.3. These components will be addressed (in details) in the following subsections.

4.3.1 XML Parsing

The input data for the tool are the extracted text in *XML* format described in § 4.2.3 and found to have varying types of *XML* tags. We used the *Python SAX* library (`xml.sax`) to parse the input text. *SAX* is a de–facto standard, rather than a formal standard, based on an

[4]in agreement with *Python* 2.x definition

original Java implementation[5]. The Simple ***A**pplication **P**rogram **I**nterface (API)* for *SAX* parsing in *Python* is a callback based system for parsing XML documents. In *SAX* parsing, the *XML* document is traversed and calls are made into a known *API* to report the occurrence of *XML* constructs (e.g. elements, text etc.) as they are encountered. This particular approach is quite popular to parse large files since SAX is event driven (memory efficient) as oppose to the ***D**ocument **O**bject **M**odel (DOM)* parsing model that converts an XML document into a parse tree and loads the whole tree in memory.

The extracted text contains large number of different tags (mostly from the *HTML* origin of the original extraction by *PDFBox*) and meta–tags. We used only a handful of those tags to normalize the text. These tags (e.g. `<div>`, `<p>`, `<b>` etc.) provided the higher level document structures (i.e. pages and paragraphs). We also used some of the basic text formatting information (i.e. bold, italic and underlined) to identify text continuity[6]. Furthermore, the files having expert annotations have a pair of special tags (e.g. `<wa_start>` and `<wa_end>`), and those were also recorded. The text content is extracted and mapped with the formatting using the character position and put into a `list` like data structure. Within every paragraph boundary, the text is further broken down into *blocks* by splitting at every `newline` character. At this stage the data structure contains the page boundaries, block boundaries and all the formatting information.

4.3.2 Redundancy Removal

The general practice in manual document generation produce some redundant text elements usually in the header and footer section of the document. These text segments often contain very little information pertaining to FP descriptions. Thus, we identified those text blocks and removed them from the data pool. We used *Dice Coefficient* (Sørensen, 1948) to quantify the similarity between two strings found in multiple pages and if the similarity is higher than a pre–defined threshold, it is considered as a candidate. There are also *two* parameters that can be changed to control the behaviour of the redundancy detection process.

Window Size (W_s) defines how many pages we search for a string S_x to be repeated. If a second occurrence of S_x is not found within W_s pages since the first occurrence, S_x will be removed from the possible candidate list. The value $W_s = 8$ was found to perform very well for our data.

[5] http://www.saxproject.org

[6] having the same format implies continuity even though the text may have been split into different blocks

Also known as the *Sørensen–Dice Coefficient*, *Dice Coefficient* is a statistic used for comparing the similarity between two sets. Like the *Jaccard Similarity Coefficient* (Jaccard, 1901), the Dice coefficient also measures set similarity but assigns twice the weight to the intersection. The original formula was,

$$QS = \frac{2C}{A+B} = \frac{2|A \cap B|}{|A| + |B|}$$

where A and B are the number of elements in *two* sets, and C is the number of elements shared by the sets. QS is the quotient of similarity and its range is $[0,1]$. A value of 0 indicates no overlap, whereas, a value of 1 indicates perfect similarity and higher numbers indicate redundancy. When taken as a string similarity measure, the coefficient may be calculated for two strings, x and y using bi–grams as follows,

$$s = \frac{2n_t}{n_x + n_y}$$

where n_t is the number of character bi–grams found in both strings, n_x is the number of bi–grams in string x and n_y is the number of bi–grams in string y.

Dice Coefficient Threshold (QS^T) defines the minimum *Dice Coefficient (QS)* two strings must score to be considered very similar, i.e. inferred to be the same. Originally the default value was set to be $QS^T = 0.8$ but it failed to detect some headers and footers (e.g. "page 37/56" and "page 58/256" results $QS = 0.777$, however, we want these to be considered as very similar), eventually, we found that $QS^T = 0.7$ performs the best for our data.

4.3.3 List Trigger Removal

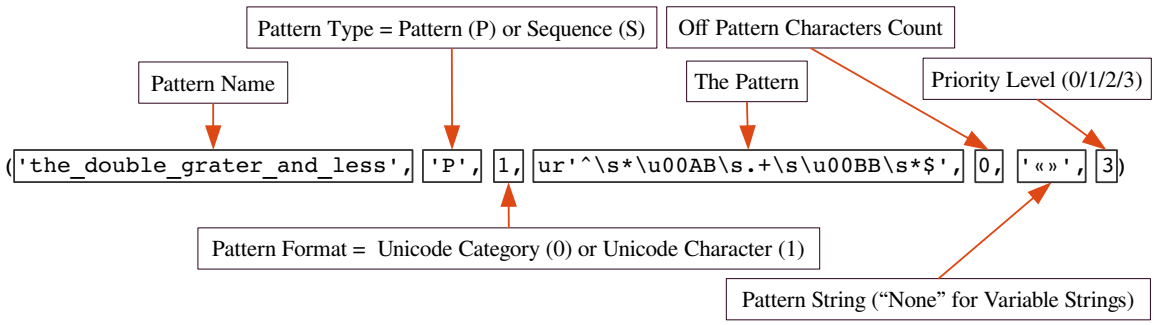
There are *two* modules that process the list like structures in the extracted text with *two* objectives: systematically identify list elements and remove the unwanted components, e.g. bullets or numbers used for list elements that do not contain any information regarding FP description. The *detection* of list triggers was achieved using a pre–defined and language specific set of ***R**egular **E**xpressions (RE)*. We also found that just defining the patterns was not enough for the task, thus we proposed a custom data structure to encompass all the necessary information. This data structure has the following features,

- There are two types of patterns: fixed patterns (e.g. "–", "« »" etc.) and sequence type patterns (e.g. "1.", "12.3.", "a)" etc.).
- each pattern can be defined to look for either unicode characters or unicode categories[7]. The difference being that a pattern looking for unicode categories requires a translation of each character sequence to its equivalent unicode category sequence before matching.

[7] http://www.fileformat.info/info/unicode/category/index.htm

- Each pattern definition also contains the number of characters at the end of each recognised string that are not a part of the required recognition pattern (if any) rather than defining the pattern boundary.
- There are three priority levels of pattern (1, 2 and 3) and in the case of multiple patterns being found in the same character sequence, only the pattern of the highest level will be recognized.
- For the fixed string pattern (e.g. "« »"), pattern definition contains a special flag.

Fig. 4.4. Example of User Definable List Detection Pattern



The patterns can be defined according to the need, especially for the adaptation of the tool to other languages. We compiled the current set of patterns for *French* by analysing our data. We are not claiming that these patterns are in anyway a complete representation of the possible list markers in a document, rather we found that except for some minor failures[8], these patterns performed exceptionally well. A significant reason for removing the list trigger patterns is to facilitate cleaner and successful parsing. We found that without these two modules, linguistic parsers have had a hard time parsing all the sentences. Furthermore, we are also recognizing the possibility of improvement on the patterns for French, for better performance.

4.3.4 Block Merging

There are *three* levels of merging performed by *three* modules in a sequential manner. The first module is the *Block Merging* module that selects eligible candidates of two subsequent *blocks* using a range of features (usually found at the end of the first string and the beginning of the second string), both as individual features and in conjunction with other features. The

[8] usually triggered by a human error e.g. a list starting with "1) 2)" is an unrecognised pattern

common features are listed in Table 4.6. For every pair of strings these features are extracted and then the merging algorithm uses them to make the decision whether to merge them or not. The string merging module uses an algorithm that is tuned to maximize merging since the text extractor originally extracted them as a single unit.

Table 4.6. The String Merging Features

ID	Reasoning
R0	The final true character (not a white space character) of the first string contains a text format (i.e. bold, italic or underlined) also present in first true character of the second String.
R1	The second string starts with a list marker.
R2	The final true character of the first string is a termination character.
R3	All the letters in the first string are title case characters.
R4	All the letters in the second string are title case characters.
R5	There are white spaces at the end of the first string or at the beginning of the second string.
R6	The final true character of the first string is a lower case letter.
R7	The first true character of the second string is a lower case letter.
R8	The first true character of the second string is a termination character.

The second module is for *Paragraph Merging*, and it determines whether the last string of a paragraph block can be merged with the first string of the immediate next paragraph block. This module uses the same sets of features but a different algorithm that is tuned to minimize merging, maintaining our confidence on the original grouping produced by the text extraction tool. This module is also responsible for merging the last string at the end of a page and the first string at the beginning of the immediate next page. Since the tokens are mapped with the information of their origin i.e. page, sentence etc., this merging only accommodates better segmentation for the parser.

The final module is the *Bracket Merging* that uses the available pairs of opening and closing brackets to merge disjoint string segments. The strings within a bracket pair are expected to belong at least to the same paragraph. A stack based matching algorithm is used that also ignores unmatched brackets. There are *three* highly customizable and user defined parameters,

Termination Character List allows the user to define possible termination characters. The default list contains the most common termination characters (e.g. “.”, “?” etc.).

Bracket Pairs allows the user to define which pair of characters shall be considered to be bracket pairs (e.g. “(” & “)”).

Bracket Distance Threshold (BD^T) defines the number of consecutive string blocks the algorithm will search for a closing bracket (C_B^x) after encountering an opening bracket (O_B^x). The default value used for our experiments was $BD^T = 5$ and if a C_B^x is not found for O_B^x within the range, O_B^x is considered to be an anomaly and removed from the search stack.

4.3.5 Data Normalization

The normalization of the data is achieved by removing all character sequences that are less probable to contain any information regarding FP description (e.g. bullets & numberings). We also removed entire blocks based on character features and simple patterns to identify improbable strings. Some of the features considered by the algorithm are listed below,

- The non alpha–numeric to alpha–numeric characters ratio should be greater than 0.2
- The number type to alphabet type characters ratio should be less than 1.0
- The string is not a *Table of Content* segment determined by a predefined pattern.
- The string is not a "fragment" i.e. a single word where none of the characters are upper case or title case characters.
- For multi word string the average word length should be greater than 2

4.3.6 Output Data Generation

The normalized text is used to generate the preprocessed data that contains two types of information: the raw text and the token list. The token list contains explicit information about the origin of a token and associated annotation information (if any). The raw text is generated using token definition that contains textual representation information for each token (e.g whether there should be a space before the token or not), producing seamless text blocks. The raw text output is targeted for the parser used for linguistic annotation and the token definition is targeted for the alignment module to combine the linguistic annotation with any pre–existing annotation in the original source (e.g. FP annotation by experts).

4.4 Linguistic Annotation

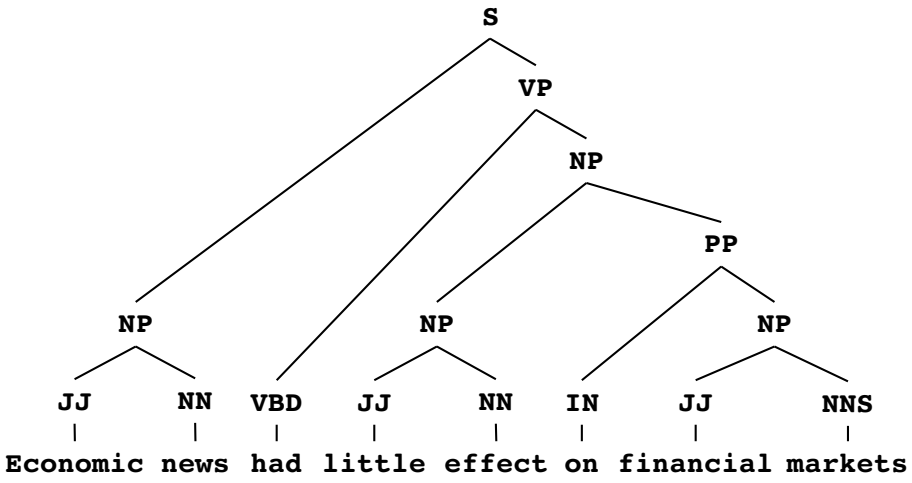
Linguistic annotation is a significant part of the corpus generation. The primary layer of linguistic information for each token of the extracted text was generated by performing

syntactic parsing of the raw text. The basic lexical annotations are lemma, detailed and fine grained ***P**arts–**o**f–**S**peech (POS)* and thus POS specific morphological features[9]. We decided to use dependency parsing that uses the principles of ***D**ependency **G**rammar (DG)*, to annotate the bi–lexical dependency relation between the tokens. We used the *Bonsai v3.2*[10], a freely available parsing tools for dependency parsing of *French* text. The output of the parsing tool is in the format specified for the multi–lingual dependency parsing shared task of *CoNLL–X* (Buchholz and Marsi, 2006) that has been adapted for the ***F**rench **T**ree-**B**ank (FTB)* (Candito et al., 2010) to accommodate the specificities of *French*. Furthermore, the tool provides in–built tokenization and sentence segmentation modules that has been used for consistent sentence definition for the corpus. In this section we shall present the theoretical aspect of the parsing, the parsing tool and parsing output statistics along with the rationales behind different choices that have been made along the way.

4.4.1 Dependency Parsing

In the modern linguistic tradition, there are two major syntactic representations for sentences. The more commonly used is the constituent based parsing that groups the words of a sentence into the possible constituent (i.e. phrase structures) and establishes the relations between the constituents in a recursive manner. Eventually, this process produces a hierarchical tree like structure where, the constituent groups build up to the sentence (see Figure 4.5). The

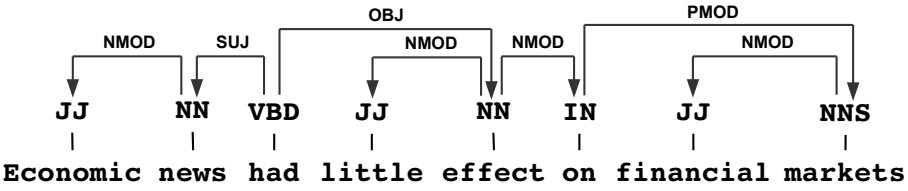
Fig. 4.5. Example of Constituent Structure for English Sentence from The Penn Treebank



[9]for nouns: number, gender etc. or for verbs: tense, mood etc.
[10]http://alpage.inria.fr/statgram/frdep/fr_stat_dep_parsing.html

Modern form of constituent based grammar is known as the ***P**hrase **S**tructure **G**rammar (PSG)* and its origin is often attributed to the work of Noam Chomsky (1957). On the other hand, dependency parsing establishes relations between each word (can also be word groups without establishing a phrasal group definition) called *dependencies* and it is often binary i.e. between two words, without establishing any phrasal nodes (see Figure 4.6[11]). The seminal

Fig. 4.6. Example of Dependency Structure for English Sentence from The Penn Treebank



work of Tesnière (1959) is usually taken as the starting point of the modern theoretical tradition of ***D**ependency **G**rammar (DG)*. The elementary notion of *dependency* is based on the idea that the syntactic structure of a sentence is a collection of binary asymmetrical relations between the words of the sentence. The terminology used by Tesnière (1959) defined *dependency* as a relation that is held between a *head* and a *dependent*. The alternative terms for head in the literatures are *governor* or *regent* for *head* i.e. *régissant* in (Tesnière, 1959) and *modifier* for *dependent* i.e. *subordonné* in (Tesnière, 1959). Historically however, DG can be traced back to the grammar of *Sanskrit* documented by Pāṇini, several centuries before the *Common Era* as, Joakim Nivre (2005) presented in the overview of DG and dependency parsing. Covington (1984) suggested that DG also has its roots in the medieval theories of grammar. However, modern dependency grammar is largely developed as a syntactic representation used by traditional grammarians, especially in Europe, and particularly in *Classical* and *Slavic* domains (Mel'čuk, 1988).

Formally, dependency parsing is based on the assumption that syntactic structure consists of lexical elements linked by binary asymmetrical relations called dependencies. DG, is the theoretical framework for dependency parsing; it is a tradition that comprises a large and a diverse family of grammatical theories and formalisms that share certain basic assumptions about syntactic structure. The set of criteria for establishing *dependency relations* and for distinguishing the *head* and the *dependent* in such relations, are of foremost importance in DG. Robinson (1970) presented four basic axioms for DG that govern the proper form of dependency structures,

[11] The English sentence for the examples (Figure 4.5 & 4.6) are taken from the *Wall Street Journal section* of the *Penn Treebank* (Marcus et al., 1994, 1993).

- One and only one element is independent.[12]
- All the other *nodes* depend directly on some other *node*.
- Each *node* can only depends directly on exactly one element.
- If *A* depends directly on *B* and some element *C* intervenes between them (i.e. in the linear order of the string), then *C* depends directly on *A* or *B* or some other intervening element.

The criteria for dependency relations are also of interest for any syntactic framework that puts some importance in the notion of *syntactic head* e.g. all the constituency–based frameworks that subscribe to some version of $\overline{X}$ *Theory* (Chomsky, 1968; Jackendoff, 1977). Below we give some of the criteria that have been proposed in the literatures (Hudson, 1990; Zwicky, 1985) for identifying a syntactic relation between a head *H* and a dependent *D* in a construction *C* as described in (Nivre, 2005):

- *H* determines the syntactic category of *C* and can often replace *C*.
- *H* determines the semantic category of *C*; *D* gives semantic specification.
- *H* is obligatory; *D* may be optional.
- *H* selects *D* and determines whether *D* is obligatory or optional.
- The form of *D* depends on *H* (agreement or government).
- The linear position of *D* is specified with reference to *H*.

The list however, is a mix of different levels of criteria: some syntactic and some semantic, that raises the question of the existence of a single coherent notion of dependency grammar. There has been some theories e.g. the concept of *head* having a prototype structure that satisfy all or most of the criteria while some peripheral instances satisfy fewer criteria (Hudson, 1990). Mel'čuk (1988), on the other hand emphasized on the idea that the word forms of a sentence can be linked by three types of dependencies: morphological, syntactic and semantic. Nikula (1986) however, argued that it is very important to distinguish between syntactic dependency in *endocentric* and *exocentric* constructions (Bloomfield, 1933). In *endocentric* construction the *head* in a dependency relation can replace the whole construction without disrupting the syntactic structure, e.g. in Figure 4.6 the *NMOD* relation holding between the noun *markets* and the adjective *financial* is an *endocentric* construction. In contrast, each component in a *exocentric* construction is equally important, i.e. the *head* cannot replace the whole construction, e.g. the *PMOD* relation holding between the preposition *on* and the noun *markets* is an exocentric construction. *Endocentric* constructions may satisfy all

[12]For example in Figure 4.6 the token (or *node*) "*had*" has no incoming dependency thus, it is the independent element. In many formal representation this node is called the *root* element and often encoded with a virtual *head* for the independent element with a *root* relation.

the aforementioned criteria but, *exocentric* constructions, by their definition, fail on the first criterion, but they may satisfy the remaining criteria.

The theoretical tradition of *DG* is united by the assumption that an essential part of the syntactic structure of sentences resides in binary asymmetrical relations holding between lexical elements. Moreover, there is a core of syntactic constructions for which the analyses given by different frameworks agree in all important respects. However, there are also important differences with respect to whether dependency analysis is assumed to exhaust syntactic analysis, and with respect to the analysis of certain types of syntactic constructions (Nivre, 2005). Our focus though is in the computational implementation of syntactic analysis based on dependency representations, i.e. representations involving lexical nodes, connected by dependency arcs, possibly labelled with dependency types. Computational implimations are often tied to a particular theory[13]. Regardless, Nivre (2005) argued that he connections between theoretical frameworks and computational systems are often indirect for dependency-based analysis, more so than for theories and parsers based on constituency analysis. He also argued that, this may be due to the relatively lower degree of formalization of dependency grammar theories in general.

Carroll (2000) distinguished two broad types of strategy for dependency parser development: grammar–driven approach and the data–driven approach, although these approaches are not mutually exclusive. For this research though, we have used a parsing system that primarily used data–driven approach. As in natural language parsing in general, the attempts to data–driven dependency parsing were grammar–driven as well, because of the use of a pre–existing corpus (usually developed using grammar–driven method) to induce the probabilistic model for disambiguation. For example, Carroll and Charniak (1992) actually used a ***P**robabilistic **C**ontext–**F**ree **G**rammar (PCFG)* model where the ***C**ontext–**F**ree **G**rammar (CFG)* is restricted to the type of DG presented by Hays (1964) and Gaifman (1965). Significant amount of research has been conducted in the field of data–driven dependency parser development and a comprehensive study is presented in (Nivre, 2005, see §3.2).

4.4.2 FTB and The Parsing Tool BONSAI

BONSAI is a set of tools and resources for the statistical dependency parsing of French. These tools and resources include, preprocessing code and pre–learnt models and grammars

[13]For example, the PLAIN system (Hellwig, 1986, 2003) which is based on ***D**ependency **U**nification **G**rammar (DUG)* or ***F**unctional **G**enerative **D**escription (FGD)* (Sgall et al., 1986) based parsing systems (Järvinen and Tapanainen, 1998; Tapanainen and Järvinen, 1997) etc.

for *three* parsers: *MaltParser*[14] (Nivre et al., 2007), ***M**inimum–**S**panning **T**ree **Parser** (MST-Parser)*[15] (McDonald et al., 2005), *Berkeley Parser*[16] (Petrov et al., 2006), and a constituent to dependencies conversion tool for French. *BONSAI* was developed under the *Project SEQUOIA*[17] (ANR–08–EMER–013), a three–year project (2009–2011) funded by the ***A**gence **N**ationale de la **R**echerche (ANR)*[18]. The official objective is listed as "Large coverage probabilistic syntactic parsing of French" thus, provides statistical parsers for French and the tools to be used for free.

Candito et al. (2010) presented the training data generation for the *BONSAI* parsers. They used the ***F**rench **T**ree–**B**ank (FTB)* (Abeillé and Barrier, 2004), a constituency–based tree–bank made up of $12,531$ sentences from the *Le Monde*[19] newspaper, for training and testing the statistical parsers. They argued that the lack of annotated data made the French statistical dependency parser development, a virtual impossibility. The *EASY Project* (Paroubek et al., 2008) has released an annotated corpus for French, containing approximately $400,000$ words of texts from various domains, such as newspaper, but also literary, medical and transcribed texts. But the annotation scheme (called *PASSAGE* formalism) mixes chunks and relations, that cannot be converted easily into full surface dependency trees usable for the training a statistical dependency parser, because it was originally developed to provide a minimum common foundation for evaluating all types of parsed output. There has been attempts to make a bi–directional converter for Passage and FTB–DEP formalism with limited success, see (Asadullah et al., 2014b). In the FTB on the other hand, each sentence is annotated with a constituent structure and words bear the following features: gender, number, mood, tense, person, definiteness, wh–feature etc., the association of a feature depends on the *POS* of course.

Candito et al. (2010) also described the *two* tree–banks automatically generated from the FTB tree–bank: *FTB–UC* and *FTB–UC–DEP*. *FTB–UC* is a modified version of the original constituency–based tree–bank, where the morphological annotation has been mapped to a simpler tag set (28 *POS* tags) , and compound structures with regular syntax are broken down into phrases containing several simple words while the rest of the sequences that were annotated as compounds in *FTB*, are merged into single tokens. In contrast, *FTB–UC–DEP* is a dependency tree–bank derived from *FTB–UC* using the classic head propagation rules,

[14] http://www.maltparser.org/
[15] http://www.seas.upenn.edu/∼strctlrn/MSTParser/MSTParser.html
[16] http://nlp.cs.berkeley.edu/
[17] https://sites.google.com/site/anrsequoia/home
[18] http://www.agence–nationale–recherche.fr/
[19] http://www.lemonde.fr/

similar to what has been proposed for English by Magerman (1995). Function labels present in the original tree–bank serve to label the corresponding dependencies. The remaining unresolved dependencies are labelled using heuristics (for dependants of non–verbal heads). Candito et al. (2010) also claimed that with this conversion technique, output dependency trees are necessarily projective[20] (see Nivre and Nilsson, 2005), and extracted dependencies are necessarily local to a phrase, i.e. the automatically converted trees can be regarded as pseudo–projective approximations to the correct dependency trees (Kahane et al., 1998). Although not conclusive, they evaluated the converted trees for 120 sentences, and report a 98% label attachment score when comparing the automatically converted dependency trees to the manually corrected ones.

Candito et al. (2010) presented the details of the development of the *BONSAI* tool and the experimental results using the *three* parsers. Although all the parsers are statistical, they are based on rather different parsing methodologies,

- ***Berkeley Parser***, a latent–variable *PCFG* parser, is a freely available implementation of the statistical training and parsing algorithms introduced in (Petrov et al., 2006; Petrov and Klein, 2007). The *Berkeley* learning algorithm uses ***E**xpectation **M**aximization (EM)* to estimate probabilities on symbols that are automatically augmented with latent annotations, a process that can be viewed as symbol splitting, as proposed by Matsuzaki et al. (2005).
- ***MSTParser*** is a freely available implementation of the parsing models described in (McDonald, 2006). These types of models are often described as *graph–based* primarily because the parsing problem is defined as searching a directed maximum spanning tree in a dense graph representation of the sentence. Graph–based parsers typically use global training algorithms, where the goal is to learn to score correct trees higher than incorrect trees.
- ***MaltParser*** is a freely available transition–based dependency parser implementation described in (Nivre, 2006, 2008). These types of models are classified as transition–based because, the strategy is to reduce the problem of parsing a sentence to the problem of finding an optimal path through an abstract transition system, or state machine.

Candito et al. (2010) argued that the choice of selecting one constituency-based parser and two different dependency parsers is based on the the work of Seddah et al. (2009). In this work, a number of constituency–based statistical parsers were evaluated on French, and the evaluation showed that the *Berkeley Parser* had significantly better performance for French than the other parsers. *Berkeley* parser has been reported to have the advantage of a strict separation of parsing model and linguistic constraints i.e. linguistic information is encoded

[20]Projectivity is a principle of tree structures by which discontinuities are identified and defined. A tree structure is said to be projective if there are no crossing dependency edges and/or projection lines (Groß, 1994).

in the treebank only, except for a language–dependent suffix list used for handling unknown words. They also reported the evaluation of the parsers (for both labelled and unlabelled dependency accuracy) along with a performance breakdown as summarized below,

- *MSTParser* showed the best labelled accuracy, whereas, *Berkeley Parser* had the best unlabelled accuracy.
- In terms of parsing runtime, *MaltParser* runs approximately 9 times faster than the *Berkeley* system, and 10 times faster than *MSTParser*, since, *MaltParser* uses a linear–time parsing algorithm, while the other two parsers have cubic time complexity.
- For nouns, *Berkeley* has the best unlabelled attachment, followed by *MSTParser* and then *MaltParser*, while for labelled attachment *Berkeley* and *MSTParser* are quite similar with *MaltParser* a bit behind.
- For prepositions, *MSTParser* is by far the best for both labelled and unlabelled attachment, with Berkeley and *MaltParser* performing equally well on unlabelled attachment and *MaltParser* performing better than *Berkeley* on labelled attachment.
- For verbs, *Berkeley* has the best performance on both labelled and unlabelled attachment, with *MSTParser* and *MaltParser* performing about equally well.
- For errors in attachment as a function of word distance, the precision and recall on dependencies of $length > 2$ tend to degrade faster for *MaltParser* than for *MSTParser* and *Berkeley*. *Berkeley* has been reported to be the most robust for dependencies of $length > 6$.
- *Berkeley* is also best at finding the correct root of sentences, while *MaltParser* often predicts more than one root for a given sentence.

For this research we choose to use the best model of the *Berkeley* available with *BONSAI* v3.2 with labelled dependency output. The decision is based on the performance of the *Berkeley* system for dependency attachment of verbs and nouns, since these are the *two* most significant *POS* associated with FP identification (see Chapter 3 § 3.3.2). Moreover, the ability of the *Berkeley* system to find correct root element and resolving long distance dependencies are particularly interesting for us since we shall be parsing noisy text which can be quite long in many instances.

4.4.3 The Data Format

BONSAI outputs the parsed text in a format defined for the *CoNLL–X* shared task for dependency parsing (Buchholz and Marsi, 2006), commonly known as *CoNLL Format*, which is an open format with the following features:

- Each sentences is separated by a blank line.

- Each sentence consists of one or more tokens, each starting on a new line.
- Each token consists of ten fields described in the Table 4.7 below.
- Fields are separated by a single tab character.
- White space or blank characters are not allowed within fields.

Table 4.7. Definition of CoNLL Format Token Fields

Column	Name	Description
1	`ID`	Token counter, starting at 1 for each new sentence (0 is reserved for the virtual *root*).
2	`FORM`	Surface form or a punctuation symbol representing the token.
3	`LEMMA`	Lemma of the surface word form, or an underscore if not available.
4	`CPOSTAG`	Coarse grained POS tag, where tag set depends on the language and the formalism.
5	`POSTAG`	Fine–grained POS tag, where the tag set depends on the language and the formalism. Can be identical to the coarse–grained POS tag (if not available).
6	`FEATS`	Unordered set of syntactic and/or morphological features (depending on the language and the formalism), separated by a vertical bar (an underscore if not available).
7	`HEAD`	Head of the current token, which is either a value of existing ID (always available for labelled outputs).
8	`DEPREL`	Name of the dependency relation to the *HEAD*. The set of dependency relations depends on the language and the formalism.
9	`PHEAD`	Projective head of current token, which is either a value of ID or zero, or an underscore if not available. The dependency structure resulting from the `PHEAD` column is guaranteed to be projective (may not be applicable for all languages and it is optional).
10	`PDEPREL`	Dependency relation to the `PHEAD` (an underscore if not available). The set of dependency relations depends on the language and the formalism.

CoNLL Format is an open format thus the features and values to be encoded can be defined to accommodate different languages and formalisms. *BONSAI* produces the output, representative of the *FTB* schema developed for the tree–bank and defines both morphological and syntactic information classes for *French*. The schema [21] was the product of the work by (Candito et al., 2010; Candito et al., 2009). One static feature is the total absence of *PHEAD* and *PDEPREL* in the output of the parsers. The schema maintained the original 13 *FTB* POS categories in the *CPOSTAG* column, whereas, used a fine–grained 28 categories for the *POSTAG* column. Distinctions such as between common and proper noun or between subordinate and coordinate conjunction has been encoded through the fine–grained categorization. There were also, 12 dependency relation defined to express the relations of verb governors, 8 relations for non–verbal governors and 8 special relations for manual annotation i.e. the systems do not produce this relations and they are only used for manual annotation.

[21] http://alpage.inria.fr/statgram/frdep/Publications/FTB-DescriptionDepSurface.pdf

4.4.4 Parsing

The *BONSAI* v3.2 contains a set of tools to parse plain text using any of the *three* parsers. We have used the *Berkeley Parser* implementation for this research. There is a simple command line script that allows the user to control the parsing behaviour using few parameters. One of the parameters is whether to use the inbuilt tokenization and sentence segmentation or not and by default, the tool set this parameter on. The other parameter is the control over the use of labelled or unlabelled dependencies, and we uses the labelled dependencies. Regardless, of the accuracy of the labels for the labelled dependency output (see Candito et al., 2010), we make use of the information that dependencies exist between *two* tokens and the dependency type. We also noticed that occasionally the parser fails to parse some sentences, primarily because of improper segmentation in the source text. During the preprocessing some paragraphs are segmented improperly e.g *one* sentence is split between *two* paragraphs. *Berkeley Parser* is a *PCFG* based parser and the *dependencies* are generated through a conversion process. Thus, if a set of token fails to translate into a proper constituent tree that set of tokens is ignored. In order to maintain robustness for our *NLP* chain, the unparsed tokens were grouped together and tagged as unparsed.

4.5 Alignment and The Corpus

Alignment and corpus generation is the last module in our NLP chain. At this stage two operations are performed: finding the alignment between the pre–tokenized output that also may contain expert annotations and the parsed output that contains lexical, morphological and syntactic annotation. Once the alignment is performed the corpus can be generated incorporating both these information. In the following subsection we shall give an overview of the alignment and corpus generation process and present our observations regarding the corpus.

4.5.1 Alignment

The alignment tool takes the parsed output and the token definition as input and produces the corpus file in a predefined custom XML format. The token definition file contains tokens that are minimum possible character groups from any text. To achieve this objective we accept only contiguous alphanumerical character span as tokens and any other type of character are

considered to be a separate token by themselves and white space characters provide context for seamless text generation. On the other hand the parser tokenizes by putting logical blocks of characters that represent linguistic unit such as, word or phrases thus, one or more tokens from the token definition (T_D) correspond to a single parser token (T_P). This relation is tested during the alignment and all the annotations associated with any T_D become associated with the corresponding T_P. formally if the tokens $T_D^a \Rightarrow \{A_a^1, A_a^2, \ldots A_a^m\}$ and $T_D^b \Rightarrow \{A_b^1, A_b^2, \ldots A_b^n\}$ found to align with T_P^x then, $T_P^x \Rightarrow \{A_a^1, A_a^2, \ldots A_a^m, A_b^1, A_b^2, \ldots A_b^n\}$. The output corpus contains the sentence structures and token definition generated by the parsed text but incorporate the expert annotation information.

The general strategy for the alignment is a basic string matching algorithm that match one character at a time from each token sequence. The tokens from each lists (i.e. T_D and T_P) is loaded in separate queues and the *top* elements are being matched until both or T_D queue is empty. If both queues are empty the next token from both lists are loaded but if only the T_D queue is empty, the next T_D token is loaded and the process is repeated until the end of file is reached. One significant consideration was the special characters of French (e.g. è, ö etc.) are normalized before the comparison since the parser tends to modify them, primarily to find correspondence in the existing dictionary i.e. for spelling agreement. The aligner also generates the corpus documents in a custom XML format thus concludes the NLP chain. The complete ***D**ocument **T**ype **D**efinition (DTD)* of the output XML files can be found in Appendix A.

4.5.2 The ProjEstimate Corpus (PEC)

The ***P**rojEstimate **C**orpus (PEC)* is a resource produced as a part of this research. It has been automatically generated and it is far from being a gold standard resource, but to the best of our knowledge it is the only such resource for FP description identification. In this section we shall provide the general statistics and significant features of the corpus. The statistics that we are interested in, are pertaining to sentences, tokens and lemmas. We are also interested in knowing how representative is the reference data (`000.ref`), primarily for associating significance to any evaluation performed using this reference. Table 4.8 listed the general statistics of the corpus with results from individual document and the combined values[22]. It also contains similar statistics extract from the reference document. One can see that in terms of number of sentences and tokens the corpus and the reference seem to be rather imbalanced

[22]The *lemma* column in the *total* row (in blue) is the total number of unique lemmas found in all the documents together and not the arithmetic sum of the lemmas found in different documents.

(41,937 to only 901 sentences). However, the number of unique lemmas is the aspect we are more interested in. Our hypothesis dictates that lemma and inter–lemma syntactic relation

Table 4.8. General Statistics of PEC and The Reference Files (000.ref)

File	Sentences	Tokens				Lemma	
		Count	Max.	Min.	Avg.	norm.	*norm. + pos*
000.ref.xml	901	9,219	74	1	10.23	1,438	1,677
001.bdf.xml	5,092	60,419	85	1	11.87	3,201	4,367
002.bdf.xml	454	7,593	72	1	16.72	1,413	1,594
003.bdf.xml	796	9,446	68	1	11.87	1,371	1,602
004.bdf.xml	1,314	21,404	71	1	16.29	1,903	2,230
005.psa.xml	5,370	51,700	76	1	9.63	2,135	3,053
006.psa.xml	408	4,450	59	1	10.91	843	984
007.psa.xml	9,805	105,553	83	1	10.77	2,208	3,246
008.psa.xml	2,021	21,752	63	1	10.76	1,179	1,504
009. psa.xml	786	7,271	117	1	9.25	1,099	1,375
010.psa.xml	1,594	15,149	87	1	9.5	1,342	1,801
011.psa.xml	663	10,217	70	1	15.41	1,029	1,344
012.psa.xml	391	3,928	68	1	10.05	681	803
013.psa.xml	1,859	15,127	52	1	8.14	1,362	1,707
014.psa.xml	3,248	50,995	99	1	15.7	1,791	2,269
015.psa.xml	1,211	11,880	53	1	9.81	722	912
016.psa.xml	753	11,283	89	1	14.98	718	913
017.psa.xml	5,296	73,303	88	1	13.84	3,315	4,489
018.psa.xml	876	11,696	84	1	13.35	1,254	1,546
Total	**41,937**	**493,166**	**–**	**–**	**11.76**	**11,250**	**16,787**

are the primary features that can be used in the search for FPs. Moreover, in close inspection we found that out of the 1,438 lemmas in the reference file 1,057 are in common with the lemmas in PEC. We also observed that many lemmas only appeared once or twice in both datasets, so we decided to remove them from the calculation process. Once removed there were 5,125 lemmas in PEC that appears more than twice and the number of lemmas found, for the same criterion, in the reference file was only 496 and of them 432 can be found in the PEC as well. Reasoning dictates that, since the reference is just a single file, many lemma may only appear once or twice regardless of the significance i.e. some specific verbs may appear just once or twice in a small file to define an action, but they might have strong tie to FPs (it has been observed, see the values in "*red*" in Table 4.9). So, when we change the

condition of the experiment to any lemma that appears in the reference and present in the PEC with a frequency higher than 2, we found that 1,021 lemmas met the criterion and now

Table 4.9. Common Lemma Statistics for PEC and 000.ref

Feature	Condition	PEC	000.ref	Common	Annotated	Common
lemma	*Unrestricted*	11,250	1,438	1,057	115	102
	$f_x > 3$	5,125	496	432		91
	$f_{PEC} > 3$		1,021	957		99
lemma + POS	*Unrestricted*	16,787	1,677	1,175	126	108
	$f_x > 3$	6,833	500	425		91
	$f_{PEC} > 3$		1,118	1,043		104

957 of them were common with the corpus. Table 4.9 summarize these findings along with the results obtained by defining the lemma in conjecture with its POS i.e. the uniqueness of the lemma in this case would be based on both the lemma and its POS (e.g. the lemma "*reference*" can be for both a "*noun*" and a "*verb*"). The *three* pruning conditions we used can be defined as follows,

- *Unrestricted:* all the lemmas are part of the statistics as long as they appear in PEC or the reference.
- $f_x > 3$: only the lemmas that appear 3 times or more in PEC and the reference respectively.
- $f_{PEC} > 3$: all the lemmas that appear 3 times or more in PEC even though they may appear less times in the reference. This criterion allows to establish a contrast of how representative the lemmas are between PEC and the reference.

We also filtered the lemmas that have been found in the annotated tokens to observe the effect of pruning the lemma list (based on both criteria i.e. $f_x > 3$ and $f_{PEC} > 3$. We found that in the normalized form (i.e. only the normalized lemma string defines an unique lemma) there were 115 lemmas and in contrast there were 126 unique lemmas when lemma and POS were used together. From Table 4.9 one can see that some annotated lemmas were missing from the PEC. So, after analyzing the data we found that there was only one annotated *verb* that was not present in the PEC and the error seem to be in the POS tagging performed by the parser (the token "*URL*" was tagged as a verb).

The other aspect of the corpus that is of interest to us is the dependency relations produced by the parser. We looked at the similarities in terms of distribution of the relations between the PEC and the reference file. The findings has been listed in Table 4.10. The relations are grouped on the basis of the interest of this research. Although *p_obj*, *de_obj* and *a_obj* are

not necessarily distinguished by most parsers we kept them in the loop for having additional information about the indirect object relations. Some of the relations e.g *ponct* and *det* (in "*red*" in the table) are of very little significance although they comprise of more than 20% of the all relations. On the other hand, the relation "*dep*" is by definition undefined and we kept it under consideration for the uncertainty involved in discarding them.

Table 4.10. The Distribution of Relations in PEC and the Reference (000.ref)

Relation	PEC		Reference		% Difference	Annotation	
	Count	Percentile	Count	Percentile		One	Both
suj	21,753	4.41	325	3.53	19.95	2	10
obj	104,076	21.1	1,906	20.67	2.04	11	88
root	41,930	8.5	901	9.77	-14.94	**53**	–
p_obj	2,113	0.43	54	0.59	-37.21	–	1
de_obj	1,706	0.35	31	0.34	2.86	–	1
a_obj	1,204	0.24	22	0.24	0	-	–
mod	109,769	22.26	2,003	21.73	2.38	8	119
ponct	63,269	12.83	1,103	11.96	6.78	8	121
dep	55,897	11.33	1,112	12.06	-6.44	2	57
det	55,378	11.23	1,009	10.94	2.58	11	65
dep_coord	11,818	2.4	264	2.86	-19.17	–	19
coord	11,181	2.27	261	2.83	-24.67	–	19
mod_rel	1,795	0.36	66	0.72	-100	–	2
aux_pass	5,829	1.18	80	0.87	26.27	–	–
ats	2,928	0.59	50	0.54	8.47	–	–
aux_tps	989	0.2	8	0.09	55	–	–
aff	774	0.16	11	0.12	25	–	–
arg	371	0.08	5	0.05	37.5	–	–
ato	228	0.05	4	0.04	20	–	–
aux_caus	67	0.01	2	0.02	-100	–	–
comp	62	0.01	2	0.02	-100	-	-
Total	**493,137**	–	**9,219**	–	–	**42**	**502**

We also explored the relation distribution in the annotated sentences and the results are interesting. Each relation contains reference to 2 tokens i.e. *source* and a *Target* and the first column list the number of relations having one of the tokens being annotated and the other column list the number of relations having both tokens annotated. One special relation is the "*root*" relation, which is in the case of this parser always equal to the number of sentences i.e.

each sentence must have one and only one *root* relation. Thus, for the annotated relations they always have just one token being annotated since the other token is a virtual *root* element. One interesting aspect was that, even though we considered the indirect relations to have significance in identifying FPs, at least in principle, the evidence showed otherwise. In conclusion, this statistics not only provide an overview of the corpus but also shapes the experimental focus and potentials of our research. The next chapter will provide the most significant aspects of our research, the experimentations performed using the corpus.

Chapter 5

Finding Function Points

This chapter is dedicated to present our contributions in terms of experiments performed for the purpose of ***F**unction **P**oints (FP)* identification. We shall also present the data analysis results and additional experiments to establish the significance of syntactic features, namely *Dependency Sub–Tree*, using a corpus from DEFT 2013 (Grouin et al., 2013), the *French* text mining evaluation workshop. There are two broad categories of experiments for FP identification that is presented here: heuristic based (using additional resources) and ***S**emi–**S**upervised **M**achine **L**earning (SSML)* based. We shall begin with the generic text analysis to understand the distribution of the data, followed by demonstrating the use of lexical semantics to accommodate FP identification and establish a baseline. Finally, we used syntactic patterns with *SSML* to improve the baseline performance. Each experimental analysis shall be presented with a brief introduction of the approach and then the experimental setup, and then the methodology and finally, the results and a brief discussion. Initially, FP identification from *two* levels were considered for this research, e.g. identification of pages and identification of sentences that contain FPs. However, we decided to evaluate only for pages containing FP descriptions due to the poor inter–annotator agreement at the sentence level in our reference data (the best was 0.66 in comparison to 1.0 for pages, see Table 4.4). Furthermore, every experimental system we have developed can be seen as a binary classification system that effectively classifies each page as *FP Positive* or *FP negative*. Thus out evaluation is parallel to the evaluation at rigidity level *L0* (introduced in § 4.1.3).

5.1 Evaluation

In this section we shall present the evaluation schema used for all the FP identification experiments. The evaluation data was a single *Functional Specification* annotated by four annotators. Following the annotation guidelines (see Appendix A § A.1), the annotation of a contiguous character span was considered to be a single annotation. However, by processing the reference file with our ***N**atural **L**anguage **P**rocessing (NLP)* chain (described in § 4.3), we produced evaluation data by translating this character level annotation into page and sentence level annotation (evaluation data) that could be used for further experimentation. Detailed annotation information (i.e. types of annotation e.g. *Transaction function, Data Function* etc.) associated with each page or sentence, is also preserved in the evaluation data. Nevertheless, to maintain accordance to the experimental setup we interpreted the existence of any annotation as *FP Positive* and the lack of any as *FP Negative*, thus evaluating the aforementioned binary classification. The *Recall*, *Precision* and *F–Measure* metric, often used in the ***I**nformation **R**etrieval (IR)* and ***I**nformation **E**xtraction (IE)* community, has been used for all the evaluation. In the following sub–sections a general overview of the metric and the evaluation system is presented.

5.1.1 Recall, Precision and F-Score

Empirical and objective evaluation plays a significant role in estimating the performance of any experimental system. When the performance of two systems or results of two experiment are compared on the same resources, usually standard statistical tests are performed, e.g. paired t-test[1], the Wilcoxon signed–rank test (Wilcoxon, 1945) or the sign test (Hull, 1993), or variants of them (Evert, 2004; Yeh, 2000). However, in the case of a binary classification problem, defined by the possible results which can be only one out of two possibilities, either of the two most popular tests (i.e. t–test and Wilcoxon signed–rank test) cannot be applied directly (Goutte and Gaussier, 2005). In contrast, binary classification evaluation is the basis for *Precision*, *Recall* (i.e. *Sensitivity*) and *F-Measure* and consequently this is a widely used metric, especially in the IR community for evaluating unranked retrieval sets. This metric can be seen as a synthetic uni–dimensional measurement that also accommodates the generation of multi–dimensional measures e.g. precision–recall curve and ***R**eceiver **O**perating **C**haracteristic (ROC)* curve see (Davis and Goadrich, 2006) for further information.

[1] originally introduced by William Sealy Gosset in 1908, see the book by Mankiewicz (2004) for the fascinating story behind the development

Precision, *recall* and *F–measure* are single valued performance measures derived from four more fundamental classes of evaluation events, e.g. *True Positive (TP)*, *False Positive (FP)*[2], *True Negative (TN)* and *False Negative (FN)*[3]. These elements can be explained the best in a binary classification scenario. The other set of concepts are *Reference* and *Hypothesis* i.e. *Reference* is the set of data points that has been considered to be ideal or desired result produced by a system or experiment, and *Hypothesis* is the set of data points produced by a system or experiment that is under scrutiny. In a binary reference set of data points there can be only one label out of two possible labels i.e. *Positive* or *Negative* (the value set can be diverse e.g. *Yes* or *No*, or *True* or *False* etc.) associated with a data point. Let us consider that the *Reference* set R that contains n data points i.e. $R = \{d_R^1,\ d_R^2\ ...\ d_R^n\}$ with the label set $l = \{P,\ N\}$. The *Hypothesis* set H thus, must have the same number of data points i.e. $H = \{d_H^1,\ d_H^2\ ...\ d_H^n\}$ using the same label set l. Now, we can associate each data point pair $(d_R^x,\ d_H^x)$: where, $1 \leq x \leq n$ to one of the cells in Table 5.1 according to the pair of labels associated with them.

Table 5.1. Frequency Map for Precision, Recall and F-Measure Calculation

		Reference	
		P	**N**
Hypothesis	**P**	*TP*	*FP*
	N	*FN*	*TN*

Now we can count the frequency of data point pairs associated with each cell of Table 5.1 and use them to calculate *Recall (r)* and *Precision (p)* using the following formulas,

$$r = \frac{TP}{TP+FN}$$
$$p = \frac{TP}{TP+FP}$$

Recall (r) thus, is the ratio of the number of correctly hypothesized *Positive* data points i.e. *TPs* to the total number of actual *Positive* data points in the *Reference* dataset i.e. *TPs* and *FNs* together. In contrast ***Precision*** is the ratio of the number of correctly hypothesized *Positive* data points i.e. *TPs* to the total number of data points hypothesized to be *Positive* i.e *TPs* and *FPs* together. A recall of 0 implies that not a single *Positive* data point has

[2] also known as Type I error
[3] also known as Type II error

been correctly hypothesized and a recall of 1 implies all the *Positive* data points are in the set of hypothesized *Positive* data points. A precision of 1 implies that all the hypothesized *Positive* data points are actual *Positive* data points in the *Reference* dataset and a precision of zero implies the same as zero recall. However, in real–world scenario, especially in IR applications, precision and recall are inversely related, i.e. as recall increases, in general precision decreases and vice versa.

The ***F-Measure***, also known as *F1* or balanced *F-Score*, is a combination of precision and recall, more specifically, it is the harmonic mean of precision and recall and expressed using the following formula,

$$F = 2 \times \frac{p \times r}{p + r}$$

it is called *F1* because it assigns equal weight to both precision and recall. If recall is weighted higher than that of the precision it is called *F2*, whereas, if precision is weighted higher it is referred to as $F_{0.5}$. These all are special cases of the generic F_β measure (e.g. for F_1 and F_2 the value of β is 1 and 2 respectively) which is based on the *effectiveness measure* by van Rijsbergen (1979), which attributes β times as much importance to recall as precision and it is defined as,

$$F_\beta = (1 + \beta^2) \times \frac{p \times r}{r + \beta^2 \times p} \text{ for, } \beta > 0$$

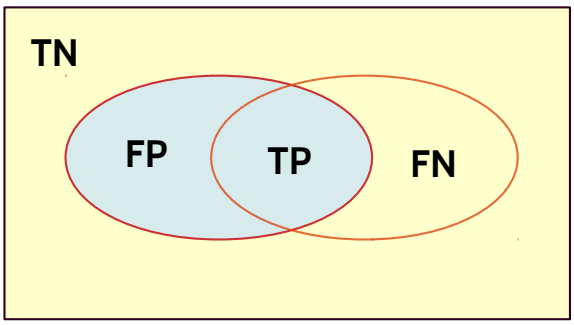
One significant aspect of this measures is the focus on the *Positive* data points only, and since, in this research we are equally interested in the performance measure in terms of *Negative* data points for each experiment. Thus, three other measures e.g. *True Negative Rate (TNR)*, also known as the *Specificity*, *Negative Predictive Value (NPV)* and *Accuracy (ACC)* were also used in the evaluation. *TNR* is the *Negative* counterpart of *Recall*, where *NPV* is the *Negative* counterpart of *Precision*, and *Accuracy* expresses fraction of classification that is correct (for both *Positive* and *Negative* data points). These measures are calculated using the same frequency map (see Table 5.1) and defined as follows,

$$TNR = \frac{TN}{TN + FP}$$
$$TNR = \frac{TN}{TN + FN}$$
$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

The performance goal of our methods is focused towards achieving a higher recall, because if all the positive pages are identified, we can construct experiments to extract specific FPs only from those pages that can accommodate detail size estimation with significant amount of

confidence in the future research. Figure 5.1 provides a visual representation of the solution space where, the ellipses represent the true data points either actual or identified by the system. The ellipse on the left represents $TP + FP$, i.e. correctly and wrongly identified

Fig. 5.1. Venn Diagram of The Solution Space



positive data points by the system. Whereas, the ellipse on the right represents data points that are actually all positive data points. The FN area represents the incorrectly identified positive data points. Higher recall reduces the area marked FN thus at least ensures that all the positive data points are correctly identified. It is a common objective at an early stage of the development of IR and IE systems as well as ours. We are trying to provide significant segmentation in the data with high confidence, although, the lack of large amount of reference data may cause drops in our systems confidence. We have acknowledged the shortcomings already, yet a very high F1 and F2 score will be welcome in terms of our understanding of the distribution of data over the PEC corpus.

5.1.2 Evaluation System

The evaluation system is designed to evaluate each experimental system on multiple sets of reference data where, each set may contain multiple annotations. For example, in our case there is only one reference document that is annotated by four annotators. The evaluation data is generated by assigning labels to each uniquely identified element e.g. pages using page numbers or sentences using unique identifier. each evaluation dataset can have multiple evaluation references (*one* for each annotator) and can be associated to a single reference data. For sentence level evaluation data extraction, we aligned the sentences extracted from each annotated version of the reference document with the reference document since, more often then not, annotators add comments along with annotation and these comments get extracted as well. Each experiment outputs in the same format as the references and thus the

evaluation tool can compare the results. While testing is performed using the reference data, the evaluation system outputs the performance of the system as a set of values for different measurement in the format presented in Table 5.2

Table 5.2. Evaluation Output Format

Measurements										
True Neg.	False Neg.	False Pos.	True Pos.	Recall	Precision	F1 Score	F2 Score	TNR	NPV	Accuracy
TN	**FN**	**FP**	**TP**	**R**	**P**	**F1**	**F2**	**SPC**	**NPV**	**ACC**

On the other hand if testing has to be performed on unlabelled data, the output only provide the number of *FP Positive* and *FP Negative* unique elements. In case of comparison between two experimental results the preference was given to the result having higher average F2–score. We chose F2 for the FP identification for pages primarily because higher recall is preferable at this stage since this stage of processing will eventually be used to complement the detail FP identification. Thus, the more positive pages we can identify will benefit the overall FP identification performance regardless of the presence of some negative pages.

5.2 Features & Resources

Feature selection is an important part of this research, especially for the ***M**achine **L**earning (ML)* experiments. The solution space of any ML experiment is fundamentally made of a set of features that can effectively map the input to the output using different learning algorithms. Thus, identifying an effective set of features can tellingly determines the success or failure of a research. For our research we have explored two sets of features,

1. Lexical and Morphological Features (Surface Form, Lemma, ***P**art **o**f **S**peech (POS)* etc.)
2. Syntactic Features (D–Grams and Dependency Sub–Trees)

We also used a set of verb lists to accommodate lexical semantic in some of the experiments, of which four were provided by industry experts and the rest were extracted from "***L**es **V**erbes **F**rançais (LVF)*" by *Jean Dubois* and *Françoise Dubois-Charlier* (Version LVF+1)[4] (Sabatier and Pesant, 2013). In the following sub–sections we shall provide details regarding the feature sets and the general description of the lexical resources.

[4]http://rali.iro.umontreal.ca/rali/?q=fr/lvf

5.2.1 Lexical and Morphological Features

The lexical and morphological features we have used are some of the commonly used features in ***N**atural **L**anguage **P**rocessing (NLP)* practice. One can often find the use of the linguistically meaningful character groups e.g. tokens (e.g. characters dimmed to belong together by a parsing system or by a set of rules) and lemmas (canonical form of a token), character group sequences e.g. *n-grams* using either tokens or lemmas, for variable length of *n* and various textual clues e.g. average token length, token or lemma frequency etc. For this research we have used the following lexical and morphological features,

1. The raw tokens defined by the parsing system
2. Lemma for each token
3. The general POS using the tag set presented in the *FTB Surface Dependency Annotation Guideline*[5] (e.g. noun, verb etc.)
4. The Specific POS i.e. POS tags that also accommodate additional morphological information as described in the *FTB Surface Dependency Annotation Guideline* (e.g. proper noun, common noun for nouns)
5. The lemma and the general POS together.

Each of these features were used in a *n-gram* configuration, where *n* is between 1 and 4. All of these features were used for the ML experiments, although only the *lemma–uni–gram* was used for the heuristic based experiments. These features were also used to generate the syntactic features, which shall be described next.

5.2.2 Syntactic Features

Dependency relations (see § 4.4.1) extracted from the parser output are the basis for our syntactic features, namely *d-gram* and *dependency sub–tree*. *D-grams* or *dependency sub–trees*, once extracted. serves as features to learn the underlying distribution of those patterns in relation to the existence of FP descriptions. Even when the minimum tree description is used i.e. only using the dependants and ignoring the dependency type, these pattern has significant advantage over a simple bi-gram model since it can capture distant relations between the dependants. Our patterns are obtained by extracting *d–grams* and *dependency sub–trees* in combination with all of the lexical and morphological feature types (e.g. lemma d–grams, specific POS dependency sub–tree etc.). Next we shall discuss the syntactic features in details.

[5] http://alpage.inria.fr/statgram/frdep/Publications/FTB-DescriptionDepSurface.pdf

D–Gram

D–grams are representations of individual dependency relations, originally intended to improve the bi–gram features by allowing distant tokens to form bi–grams, considering such dependency exists. *D–grams* are formed from the dependency tree generated by the dependency parsers. The use of *d–grams* for text representation that was used for *Sentiment Analysis* task, has already been demonstrated by Pak and Paroubek (2011). They defined *d–grams*, a concept loosely related to *skip–gram* (Guthrie et al., 2006), as a tuple of one dependency relation and two dependants i.e. the *source* and the *target* of the relation. Formally, a *d–gram* is as follows,

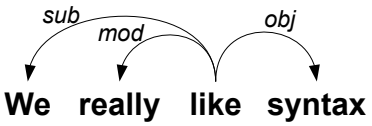
$$d-gram = (\ e^{src},\ e^{rel},\ e^{target}\ )$$

d–grams are generated at sentence level, thus one set of *d–grams* is generated for each sentence. Given a sentence S having k dependency relations, the *d–gram* set is as follows,

$$d-gram(S_i) = \{(e^s_1, e^d_1, e^t_1), (e^s_2, e^d_2, e^t_2), \ldots (e^s_k, e^d_k, e^t_k)\}$$

For an example, let us consider the following sentence,

Fig. 5.2. Dependency Parser Output: As An Example for *D–Gram* Generation



Thus, the resulting *d–grams* from this sentence are as follows,

$$(\ like,\ sub,\ We\ ) \qquad (\ like,\ mod,\ really\ ) \qquad (\ like,\ obj,\ syntax\ )$$

Furthermore, to achieve more generalization in the features, Pak and Paroubek (2011) also introduced the notion of using *wildcards* in the tuple that would substitute any token or relation for the given *wildcard* position. With the introduction of one *wildcard* (denoted as *d–gram*$*$) the following patterns can be generated for each original *d–gram*,

$$d-gram* = \begin{cases} (\ e^{src},\ e^{rel},\ *\ ) \\ (\ *,\ e^{rel},\ e^{target}\ ) \\ (\ e^{src},\ *,\ e^{target}\ ) \end{cases}$$

From the previous example (Figure 5.2), the following *d–gram*$*$ patterns shall be produced,

$$
\begin{array}{lll}
(*, sub, We) & (like, *, We) & (like, sub, *) \\
(*, mod, really) & (like, *, really) & (like, mod, *) \\
(*, obj, syntax) & (like, *, syntax) & (like, obj, *)
\end{array}
$$

Pak and Paroubek (2011) also presented a variation of the aforementioned pattern by replacing two elements in a *d–gram* with *wildcards*. They aptly named them extended *d–gram* (*xd–gram*) and the following patterns can be generated from the original *d–gram* pattern,

$$
xd-gram = \begin{cases} (e^{src}, *, *) \\ (*, *, e^{target}) \end{cases}
$$

Pak and Paroubek (2011) however, refrained from removing both *source* and *target* since it does not make much sense and in agreement with them, we are operating under the assumption that tokens carry the information and dependency relation type augment that information to generate information rich patterns.They also reported an interesting observation that the first of these patterns is effectively a *uni–gram* representation of the *source*, whereas the second pattern also represents the instances where d_{target} is the *head* of a relation. Using the same example (Figure 5.2), the following unique *xd–gram* patterns shall be produced,

$$
\begin{array}{ll}
(*, *, We) & (like, *, *) \\
(*, *, really) & (*, *, syntax)
\end{array}
$$

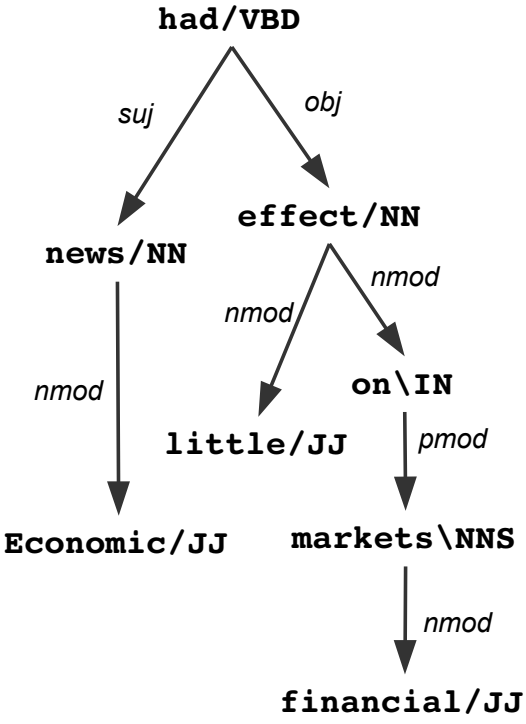
We adopted the *d–gram* representation and generated features for our ML based experiments. The fundamental difference between our use of the *d–gram* patterns with that of the application by Pak and Paroubek (2011) are twofold. Firstly, instead of just using tokens or lemmas to generate *d–grams* we used all of the five lexical and morphological features presented in § 5.2.1 to generate a total of 15 feature spaces. And finally, they used a method described as "*fusing and pruning of triples*", where, *fusing* is the combination of a *target* and a *source* if the type of dependency convey more information relevant to a specific task. Afterwards, the fused node is used for any other relation that may refer to any of original nodes that has been fused. As an example, the authors (Pak and Paroubek, 2011) reported that the *negation* relations were fused since, the relation is more relevant to the sentiment analysis task. We did not perform any fusing at this stage of our research. *Pruning* on the other hand is the removal

of relations that are less relevant, e.g. the authors reported to remove any *noun modifiers* since thy deemed them less informative to their model. We however, used pruning a little differently, e.g. we removed all the patterns having any *stop words* for French.

Dependency Sub–Tree

Dependency sub–tree patterns can be seen as an extension of the *d–gram* patterns. These patterns use the interconnected tokens (by some dependency relation) from a dependency tree and each pattern is a sub–tree of a branch of that tree. Sub–tree patterns are also extracted at sentence level, thus each sentence produces a set of sub–trees. Furthermore, for each *sub–tree* of length greater than 2 (i.e. more than one dependency relation) there will also be sub–trees of all possible intermediate integer lengths. Formally, if there exist a sub–tree of length k where $k > 2$, there will be $(k-1)$ sub–trees of length $\{2,3,4,\ldots(k-1),k\}$. Let us consider the following example,

Fig. 5.3. Dependency Parser Output: As An Example for *Dependency Sub–Tree* Generation



Each token in the example (e.g. had/VBD) is a node of the tree and traversal of this tree to extract sub–tree patterns can be performed in either direction. For out experiments however, we traverse each tree on the directed path output by the parser (i.e. *source* to *target*). Some

of the patterns that shall be extracted are as follows,

$$had \rightarrow news$$
$$news \rightarrow Economic$$
$$had \rightarrow news \rightarrow Economic$$
$$had \rightarrow effect$$
$$effect \rightarrow little$$
$$had \rightarrow effect \rightarrow on \rightarrow market \rightarrow finance$$

$$had/VBD \rightarrow effect/NN \rightarrow little/JJ$$
$$effect/NN \rightarrow on/IN$$
$$had/VBD \rightarrow effect/NN \rightarrow on/IN$$
$$on/IN \rightarrow market/NNS \rightarrow finance/JJ$$

These patterns were extracted automatically as in (Nouvel et al., 2014). Although in the above example we only presented either the token or the token and its POS together as a node, all the lexical and morphological features found in § 5.2.1 have been used for experiments. Furthermore, patterns of length 2 are equivalent to the *d–gram∗* pattern, (e^{src}, $*$, e^{target}), e.g. $had \rightarrow news$. The general characteristics of the patterns we extracted can be summarized as follows,

1. Each element is a sequence of tokens and/or their morphological features
2. The dependency relations are discarded in the pattern but used for pruning less useful dependency relations.
3. All possible tree depth can be explored, but patterns longer then 5 were pruned out.
4. Elements of a pattern can represent all the lexical and morphological features (see § 5.2.1).
5. Each element of a given pattern is concrete i.e. unlike *d–gram∗* or *xd–gram*, variable elements do not exist.

One interesting observation was that since our focus was primarily with *verbs* and specific set of relations (e.g. subject, direct object etc.), we have patterns where one of the *source* or the *target* is a *verb* (more often than not it is the *target*). We have effectively, implemented a pruning algorithm to extract only those patterns, however, we did not use them in our experiments since satisfactory results were obtained without using pruning. We used the sub–tree patterns in the ML experiments for FP identification and on a separate experiment

with *cooking recipes*, where several supervised learning methods were used. We found that sub–tree patterns are powerful features to be used in supervised learning setup, so we used them in our semi–supervised learning experiments as well. In the cooking recipe experiment, we normalized the *coordination* relations but without any significant gain in performance (see § 5.4.2), so we decided not to normalize the *coordination* relations for the FP identification task.

5.2.3 Lexical Resources

We have used a set of verb list provided by the FP counting professionals, to accommodate lexical semantic as a part of our *Heuristics* based experiments. They listed the verbs on the premise that they are specific to FP descriptions or have domain specific semantics[6]. Furthermore, some lists also provided the association of each verb with the broader FP groups (i.e. *Transaction Function* and *Data Function*). We had four lists from the experts as described in Table 5.3,

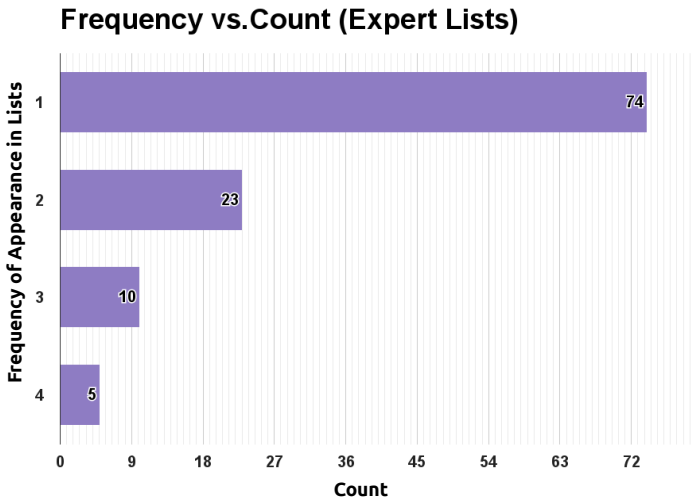
Table 5.3. General Statistics of the Expert Lists

Designation	Transaction Function	Data Function	Undefined	Total
ACAPI	49	–	–	49
BdF	34	–	–	34
OPI	35	1	–	36
PSA	15	–	40	55

Each verb was presented in the *infinitive* form and the lists have been manually treated to improve consistency in terms of proper spelling, accents etc. Some of the elements were multi–token constructions rather than unit lemmas and we decided to discard them for the initial set of experiments. We have also constructed a combined verb list using the four expert lists, where instead of associating each verb with possible FP type (since the association is inconsistent across the lists and except for one verb they are either associated with *Transaction Functions* or left without any association), we counted the frequency of appearance in terms of the number of lists they were found. There were, 112 verbs in the combined list and their list appearance frequency is presented in the following illustration,

[6] e.g. *créer* (to create) often found in the context of a *Transaction Function*, more specifically, in the context of an External Input.

Fig. 5.4. List Appearance Frequency of Expert Listed Verbs



We have used this frequencies to associate significance of a verb (from the lists) in the domain and used them as weighting factor in the *linear combination* experiments. The exact conditions under which these lists were compiled were not provided by the partner organizations. We do not know whether the authors have shared their lists with each other or if they have used the same set of source documents nor whether the source documents are also included in our dataset. The presence of organizational bias is also considered and thus, we have relative confidence on the quality and representativeness of these lists. However, the performance of these lists in the *Heuristic* based experiments was a contributing factor, for us to include them in our research.

We have also extracted few lists from the "***L**es **V**erbes **F**rançais (LVF)*" by *Jean Dubois* and *Françoise Dubois-Charlier* (Version LVF+1) (Sabatier and Pesant, 2013). It is a freely available French lexical resource that provides classifications based on domain, schema and syntactic and semantic classes for French verbs. We have developed lists for the *Informatics* domain (`INF`) and sub–classes of the verbs associated with movement (to maintain a parallel to the FP concept). In the following section we shall present the other aspect regarding the experimental prerequisite, especially for ML experiments: generation of data points.

5.3 Data Point Generation & Analysis

Data point generation and primary analysis is a part of the experimental setup for the *Semi–Supervised ML* method applied in our research. Data point definition and generation is

necessary for defining the search space where the ML algorithm may be applied. Data points were generated for each feature set described in § 5.2 and there are two aspects of data point analysis, *Vectorization* i.e. representation of each data point as a multidimensional vector and ***P**rincipal **C**omponent **A**nalysis (PCA)* i.e. analysis of the data points at more significant but lower dimension for learning and visualization. To adapt to the classification methods we applied in our research, the task definition was developed as follows,

1. We want each page to be classified as *FP Positive* or *FP Negative*.
2. Each page is considered to be a document.
3. Each page has a unique identifier and all the features given a feature type found in a page are used as raw input.
4. The page matrix (equivalent to *Document Matrix* in document classification task) represents each page as a row and all possible unique features as columns (see Table 5.4).
5. The intersection of each row and column contains a value calculated based on the vectorization method being used.
6. The page matrix is the search space where the ML algorithms shall be applied.

Table 5.4. General Representation of The Document Matrix

	"***go***", "***have***", "***system***" etc. for ***lemma***			
	f_1	f_2	$\cdots$	f_n
P_1	$v_{(1,1)}$	$v_{(1,2)}$	$\cdots$	$v_{(1,n)}$
P_2	$v_{(2,1)}$	$v_{(2,2)}$	$\cdots$	$v_{(2,n)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
P_m	$v_{(m,1)}$	$v_{(m,2)}$	$\cdots$	$v_{(m,n)}$

In recent years, an advanced form vectorization has been used in document and topic modelling extensively that is often referred to as *Lexical Embedding* (Wang et al., 2015). However, we used the traditional vectorization techniques (e.g. frequency, *TF–IDF* etc.), keeping the future prospect of using other vectorization methods open. We have also used *Principal Component Analysis (PCA)* to reduce the dimension of the search space as low as just 1. The Original rationale was to find a balance between the reduction of the search space (i.e. faster processing) and preserving maximum information content (i.e. significant separation of data) for the ML algorithms. However during initial experimentation, we observed that the performance was significantly better at lower dimensions. The following subsections will introduce the vectorization techniques and PCA methods used in our research.

5.3.1 Vectorization

We have used three types of vectorization for experiments: *Frequency*, *Hash Value* and *TF-IDF*. Frequency is the raw count of each feature found for a given element in the target domain. Hash value on the other hand converts each feature into a integer using hashing techniques. This particular vectorization is highly memory efficient and only produce hash values that can cause *Collision* problem i.e. assignment of the same value to two different features for a feature size of 2^{18} or higher which is sufficient enough for our experiments. One significant problem with this vectorization is that once hashed it is not possible to revert back to the original feature string. Thus this vectorization has been used to cover all the basics rather than a decisive method. *TF–IDF*. *TF–IDF* is a significant metric often used to generate the search space and to identify the underlying distribution in that search space. We would apply it for visual data analysis and ML experiments. The fundamentals and necessary background has been addressed in § 2.1.3. We however, used the metric in a different manner, i.e. instead of calculating *TF–IDF* for all the pages of multiple documents together, we calculated the *TF–IDF* page matrix for each document separately and then mapped it to the complete feature set generated from all the documents together. We are calling it "*Localized TF–IDF*" and once all the *TF–IDF* page matrix are generated, they are concatenated to produce a single search space. Since, we have designed our experiments to focus on common elements in the descriptions of FPs, distribution of search terms were necessary to be calculated on a local basis. The premise of our hypothesis dictates that local distribution of features associated with FPs remain within an acceptable standard deviation of the mean for all specification documents. Thus, using a general *TF–IDF* page matrix is contradictory to our original assumptions. Furthermore, in real world situations, the prediction model will eventually be used for each specification separately, thus the model will fail to achieve equivalent range on a single document if general *TF–IDF* page matrix is used for the training of such models.

5.3.2 Principal Component Analysis (PCA)

PCA is a traditional family of methods to reduce the dimension of the data, analyse and visualize them for the identification of existing patterns and selecting parameters. In standard PCA the multidimensional data is reduced to its n principal components to easily visualize the hidden patterns with the expectation that the same patterns remain prominent in the higher order data representations. The *Python* implementation of PCA used in our experiments are

the classic PCA and ***S**ingular **V**alue **D**ecomposition (SVD)*, however we finally, decided to use only SVD because of the inability of the classic PCA implementation (in *scikit–learn*[7]) to handle sparse data since our data points are highly sparse. Both the classic PCA and SVD have been discussed in § 2.1.3 for a clearer picture of our decision making process. We shall investigate the distribution of our data and as the search space for the semi–supervised learning using SVD as the primary PCA method.

5.3.3 Visual Data Point Analysis

Visual inspection of the distribution of the data was performed using the first two primary components of the document matrix produced from the data. In this section we shall discuss some of the interesting observations regarding the two–dimensional distribution of the data. The experimental setup that has been used, is the same as the one we used for the ML experiments. There are two groups of data presented in each graph, the training data which is the page matrix generated from the 18 specifications from project partners and the page matrix of the reference data (`ref000`, see §) where, the positive and negative examples are plotted distinguishably. The objective is to observe range of the distribution and possible pre–existing cluster configurations.

The data for the experiment has been grouped as datasets, e.g. the 18 specifications comprises the training dataset and 1 reference specification is the reference dataset. Each dataset is then used to produce the page matrix using *local TF–IDF* vectorization (see § 5.3.1). PCA (SVD in our case) is then applied to each page matrix to reduce the dimension to two primary components. The training dataset is then plotted in a two dimensional *Euclidean* space, plotting the first principal component on the horizontal axis and the second principal component on the vertical axis. The training dataset is plotted in *grey* and using small dots for each data point. The reference data set is then plotted using larger dots and in *blue* for the positive data points and in *red* for negative data points. Some of the generated plots for the surface form and lemma in lexical and syntactic feature configurations are presented next.

The evaluation data used for the plotting is from the annotator designated *BdF_MG*. this particular annotator annotated the maximum number of pages as positive (11) and all the other annotators assigned positive values to some subset of those 11 pages. One important observation is that six of those pages were marked by all the annotators as positive. Thus, the plots are representative of all the annotations for the reference dataset. We actually attempted

[7] http://scikit-learn.org/stable/

Semi–Supervised learning with complete page matrix and only the first primary component and found better performance from the later, thus instigated the visual analysis of the data to identify pre–existing and prominent separation in the data at lower dimensions. We were also interested to identify a reasonable cluster number, to have a balance between increasing training efficiency (i.e. time complexity reduction) and reducing possible over-fitting. Visual observation helps to identify a range of values with which we can use to find the best cluster count.

Fig. 5.5. 2D Scatter for `surface form` : `uni-gram` | `bi-gram`

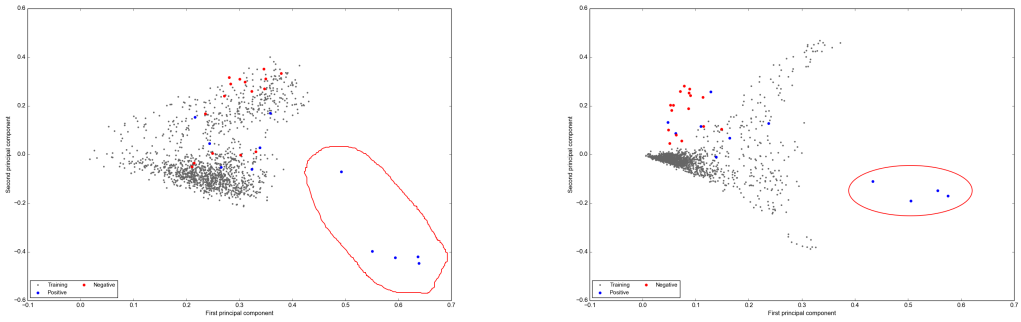
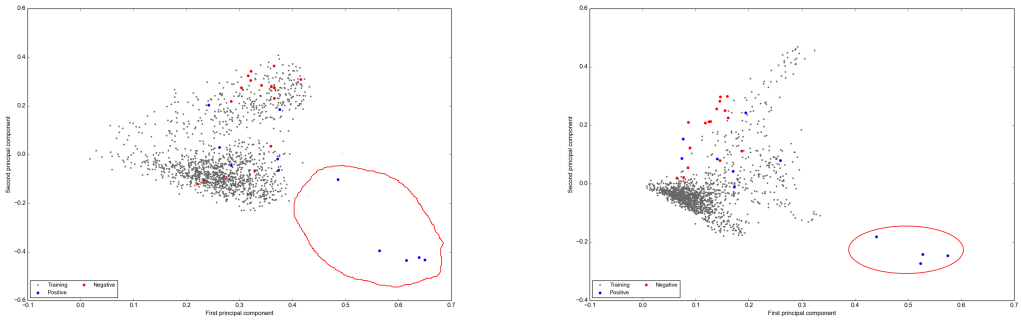


Fig. 5.6. 2D Scatter for `lemma` : `uni-gram` | `bi-gram`



In our analysis we found that in the feature space (both lexical and syntactic) generated using the lexical features (surface form and lemma), some positive examples (between 3 and 5 data points) are out of the range in contrast to the training data points. Furthermore, the pages that were out of range were the pages that has been annotated by all the annotators. This pattern however, was not observed in the feature sets using morphological features. We conclude that, the reason for such pattern to occur, may contribute to the nature of the specification (at least those pages) being different from the specification used for training. We decided to

Fig. 5.7. 2D Scatter for `surface form` : `d-gram` | `d-gram*`

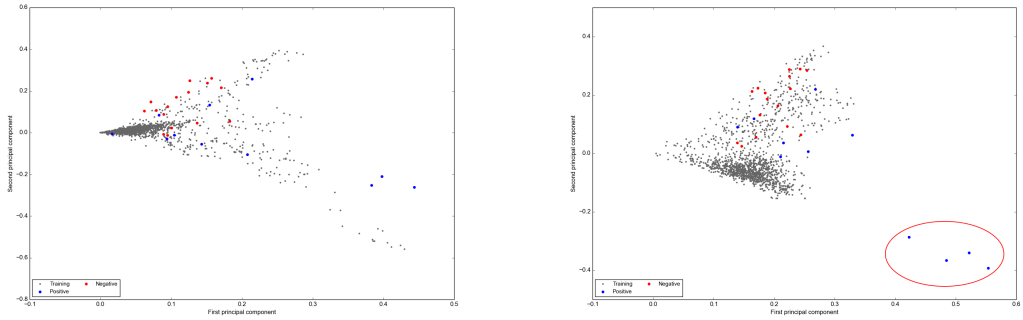


Fig. 5.8. 2D Scatter for `lemma` : `d-gram` | `d-gram*`

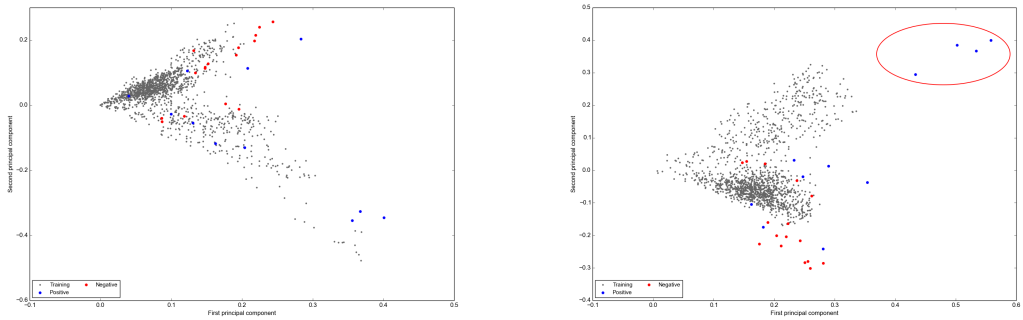
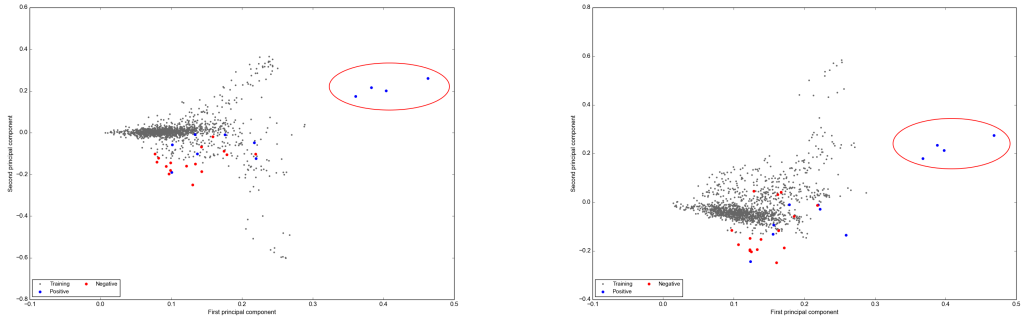


Fig. 5.9. 2D Scatter for `surface form` | `lemma` : `dependency sub-tree`



run the ML algorithms in lower dimension ranges (between 1 and 5) to test the performance and conclusively determine the best dimension. We were also unable to observe pre–existing cluster or distinct separation in the data at 2 dimensions. It was clear though, that we need a large number of clusters to segregate the data properly. Thus we decided to use large cluster range (between 7 and 15) for the ML experiments. In the next section we shall present the different experiments performed in our research.

5.4 Experiments

There are three sets of experiments that are presented in the next subsections. The first set is the *Heuristic Based* experiments, where we used string matching to identify *FP Positive* pages and also two variations of liner combination based methods. Next is a separate set of experiments to identify ingredients from cooking recipes, where we tried to evaluate the performance of syntactic features, especially *Dependency Sub–Tree Patterns* using a large labelled dataset, objectively. The final set of experiments used *k–means Unsupervised Clustering* to design a *Semi–Supervised* learning method using all feature sets for FP identification. Each set of experiments shall be presented in the next sub–sections and we shall provide any theoretical basis necessary for the experiments, followed by the experimental setup that has been used , the data and the features and finally the results with a brief discussion on our findings.

5.4.1 Heuristic Methods

The *Heuristic Methods* were applied in the reference data and evaluated against the labels associated for each page. There are two experimental setups that have been used, *String Matching* and *Linear Combination*. All of these experiments were performed using the expert verb lists (see § 5.2.3), although in different capacities. Both types of experiments also used the lexical resources (described in § 5.2.3) and classify each page as *FP Positive* or *FP Negative*. Furthermore, among the feature sets only the lemma was used for these experiments. Next subsections will describe these experiments in detail.

String Matching

This is the simplest form of the lexical experiments. It used the lemma feature space, which represents each page as a sequence of lemmas. The algorithm takes one expert list and count the frequency of any verb from the list in a page. A threshold value is then used to determine the class (FP Positive or FP Negative) for each page. The frequency has been normalized to remain within the range of 0 and 1 using the following formula,

$$z_i = \frac{f_i - minimum(f) + \alpha}{maximum(f) - minimum(f)}$$

The formula is representative of the calculation of *z–score* in statistics and α is the normalizing factor to eliminate the possibility of having 0 as numerator. However, it may produce values grater than 1.0, thus, we capped the maximum possible normalized value to 1.0. For evaluation we tested thresholds between 0.0 and 1.0 with an interval of 0.01. Although we evaluated the performance based on average *F2* for all the annotators, we were also interested in low numbers for both *False Positives* and *False Negatives*. Among the lists "*PSA*" produced the best overall result in that respect. The complete result is presented below.

Table 5.5. String Matching (`list = ''PSA'', threshold = 0.05`)

ANNOT	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	10	6	20	**0**	1.000	0.625	0.769	0.893	0.769	1.00	0.833
BdF_DB	6	10	20	**0**	1.000	0.375	0.545	0.750	0.667	1.00	0.722
BdF_MG	10	6	19	**1**	0.909	0.625	0.741	0.833	0.760	0.95	0.806
PSA	6	10	20	**0**	1.000	0.375	0.545	0.750	0.667	1.00	0.722

The evaluation was performed considering the total pages to be 30 since the other 6 pages were ignored during feature generation and the criterion used for selecting pages was having more than 5 sentences. The best performing list is impressive in terms of reducing the search space from 37 pages to 16 pages thus reducing reading time. Although we cannot claim the performance to be conclusive considering the limited evaluation data, it is at least promising. The high performance at a low threshold can be attributed to the fact that some of the positive pages contain very few occurrences of the lexicons from the lists and some of them only contain functional group or external application annotation.

Linear Combination

Liner combination in mathematics, is an expression constructed from a set of terms by multiplying each term by a constant and adding the results, e.g. a linear combination of x and y would be any expression of the form $ax + by$, where a and b are constants (Lay et al., 2015). The definition has been adopted for our experiment where $F = \{f_1, f_2, f_3, \ldots f_n\}$ is the frequency vector for a set of n lemmas (form a list of lexicons) and $W = \{w_1, w_2, w_3, \ldots w_n\}$ is the corresponding weight vector. Thus, the liner combination has been defined as,

$$\sum_{i=1}^{n} f_i \times w_i$$

The dataset used for this experiment is again the lemma feature set, where each page is defined as a set of lemmas. For the lexical resource we used, the combined lexicon list produced from the four expert list. This list contains non-redundant set of lemmas and the number of lists where they were found. The weight vector was generated by normalizing (described for string matching) the frequency of each lemma in terms of appearance in the expert lists. For each page the frequency vector is generated based on the frequency of each lemma from the combined lexicon list. Finally the linear combination is used to generate a single real number that is then normalized (*z–score*) to calculate the final score for each page. Based on a pre–defined threshold, each page is classified as *FP Positive* or *FP Negative*. We have found that threshold 0.6 performed the best. The complete result is presented in Table 5.6.

Table 5.6. Linear Combination (`threshold = 0.6`)

ANNOT	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	10	7	19	**0**	1.000	0.588	0.741	0.877	0.731	1	0.806
BdF_DB	6	11	19	**0**	1.000	0.353	0.522	0.732	0.633	1	0.694
BdF_MG	11	6	19	**0**	1.000	0.647	0.786	0.902	0.760	1	0.833
PSA	6	11	19	**0**	1.000	0.353	0.522	0.732	0.633	1	0.694

This method effectively used direct string matching as in the previous experiment to construct the frequency vector. An important observation was the lack of the significant presence of the lexicon from the list in many of the positive pages that results in better performance for low threshold. Although, This system failed to outperform even the baseline system, in

terms of number of *False Positives* we can consider the performance to coincide with our experimental goals. Next we shall present a variant of this experiment.

Linear Combination with Learned Weight

This experimental setup is equivalent to the previous setup with two significant differences. First of all, we are using each of the expert lists and the combined list, and we are now learning the weight of each lemma for a given list from the training dataset. The frequency of each lemma is counted from the training data then normalized (*z–score*) to generate the weight matrix. Then the frequency matrix is generated for each page, using the weight matrix and the liner combination from the previous experiment, a single real number is assigned. Again using a pre–defined threshold each page is classified as *FP Positive* or *FP Negative*.

In our experiment the performance improvement was observed for three lists: *PSA*, *OPI* and *BdF* with different yet rather close thresholds (0.11, 0.14 and 0.12 respectively). We would like to acknowledge that we accept the significant bias the lists pose and their effect in the baseline performance. We have relative confidence in the results considering the small amount of evaluation data we have, however, under the circumstances these performance is a promising methodology to adopt given improved resources can be developed. The complete result is presented in the Table 5.7.

One can see that in terms of *recall* the *PSA* and *OPI* are exactly the same, however, the *precision* is better for *PSA* and it is an improvement in terms of our research goals since it is effectively and correctly reducing the total pages to read down to 14. The performance is also an improvement over the baseline system by reducing *FP Positive* pages by 2 on the other hand the system that used the *BdF* list produced a perfect recall. Although, with respect to the baseline system the total *FP Positive* pages are higher by 2 pages. The research goals i.e. identification of pages with FP certainly best fit the results obtained with the *BdF* list. Again, the low threshold can be contributed to some of the *FP Positive* pages having very few lemmas from the lists. Our next experiments will attempt to reduce total positive pages further without compromising recall too much using syntactic features and ML methods. We are inclined towards the syntactic features to established our hypothesis and to remove the bias posed by the expert lists. But first the next experiment will try to establish the effectiveness of the syntactic features, especially *Dependency Sub–Tree*, objectively.

Table 5.7. Linear Combination (Learned Weight)

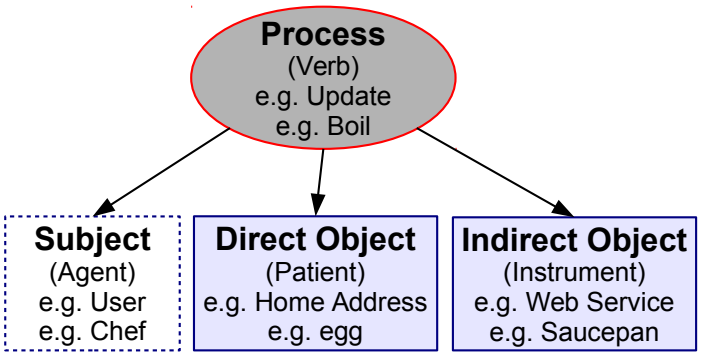
ANNOT	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
			`list = "PSA", threshold = 0.11`								
Acapi_JNV	10	4	21	**1**	0.909	0.714	0.800	0.862	0.840	0.955	0.861
BdF_DB	6	8	22	**0**	1.000	0.429	0.600	0.789	0.733	1.000	0.778
BdF_MG	10	4	22	**0**	1.000	0.714	0.833	0.926	0.846	1.000	0.889
PSA	6	8	22	**0**	1.000	0.429	0.600	0.789	0.733	1.000	0.778
			`list = "OPI", threshold = 0.14`								
Acapi_JNV	10	5	20	**1**	0.909	0.667	0.769	0.847	0.800	0.952	0.833
BdF_DB	6	9	21	**0**	1.000	0.400	0.571	0.769	0.700	1.000	0.750
BdF_MG	10	5	21	**0**	1.000	0.667	0.800	0.909	0.808	1.000	0.861
PSA	6	9	21	**0**	1.000	0.400	0.571	0.769	0.700	1.000	0.750
			`list = "BdF", threshold = 0.12`								
Acapi_JNV	11	7	18	**0**	1.000	0.611	0.759	0.887	0.720	1.000	0.806
BdF_DB	6	12	18	**0**	1.000	0.333	0.500	0.714	0.600	1.000	0.667
BdF_MG	10	8	18	**0**	1.000	0.556	0.714	0.862	0.692	1.000	0.778
PSA	6	12	18	**0**	1.000	0.333	0.500	0.714	0.600	1.000	0.667

5.4.2 Ingredient Extraction from Cooking Recipes

The *Heuristic* experiments demonstrated the ability to identify FPs in a page using lexical feature, namely *lemmas*. Our original hypothesis however, was that syntactic features can outperform lexical features. Furthermore, the lemmas used in the *Heuristic* experiments as lexical resource can have possible bias towards the annotation in the reference data due to our lack of knowledge of the origin of these resources and the possible a priori knowledge of the reference data by the resource developers. Syntactic features, on the other hand are intended to be used for *Semi–Supervised* ML algorithms, thus effectively remove the possibility of biased lexical resources to effect the research objectives with subjectivity. We are specially interested to use *Dependency Sub–Tree* as a feature but due to the lack of large FP evaluation data it was difficult to establish its effectiveness objectively.

We decided to use *Dependency Sub–Tree* as a feature with large amount of data, especially evaluation data from a domain that is not too dissimilar to that of software specifications. We had a large amount of data in the form of *Cooking Recipes* form a text mining evaluation campaign (*DEFT 2013*) including large amount of gold standard evaluation data. We are also interested in the generic *Verb–Noun* interaction in specification documents as we found that it has already been used successfully by Faure and Nédellec (1999) to extract *Sub–Categorization Frames* of verbs and ontologies from the syntactic parsing of technical texts in natural language. Cooking recipes are very mush within the ontological definition of specifications and we have found the existence of the *Verb–Noun* interaction patterns expected in a specification documents as illustrated in Figure 5.10.

Fig. 5.10. Generic Verb–Noun Interaction Pattern



In Figure 5.10 we presented a generic frame with semantic descriptions, however, we are analysing all the data from the syntactic dependency representation only. The *DEFT* task was to identify ingredient from a given cooking recipe, which is different from our FP

identification task. From the generic model (Figure 5.10), for the ingredient extraction task the object of interest is the *direct object* relations whereas, for the FP identification task, the whole *Verb–Noun* pattern frame is useful. We are interested to use the *Dependency Sub–Tree* patterns that can capture the information in the patterns and use ML algorithms to identify patterns related to a given task objective. For the ingredient extraction task we have used *Supervised* ML methods, motivated by the availability of a large training corpus. We designed our method to work with cooking recipes written in *French*, although, the adaptation of the method for other domain or languages depends only on the use of a language specific (if necessary, also domain specific) dependency parser. The performance of our method on the *DEFT2013* data set (Grouin et al., 2013) is quite satisfactory since it significantly outperforms the best performing system from the original challenge (0.75 vs 0.66 MAP).

The DEFT Challenge: 2013

The *DEFT Challenge* is an annual French text mining evaluation workshop. Inspired by the Computer Cooking Contest[8], the 9th edition of this challenge, titled *DEFT 2013* (Grouin et al., 2013), was focused on the analysis of recipes written in French. The challenge a large set of cooking recipes in French for the participants to develop their systems with. The details of the *DEFT* corpus is presented in table 5.8.

Table 5.8. DEFT Corpus

Corpus	Recipes	Sentences	Words	Ingredients
Training	13,866	141,613	2,013,934	101,563
Test	9,230	93,338	1,311,802	74,796

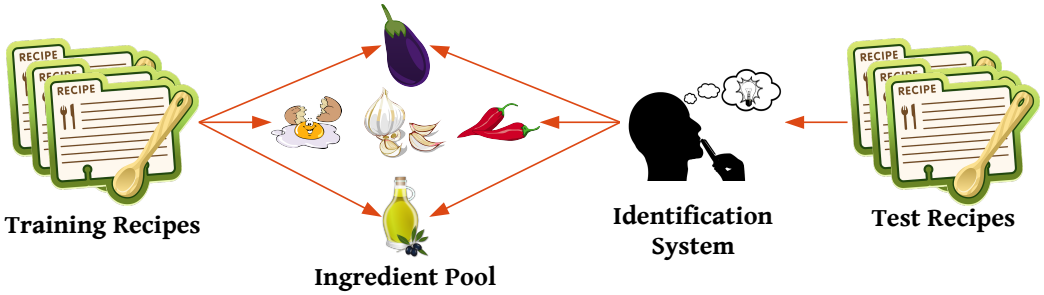
The challenge was designed with 4 tasks from 2 main categories,

1. Document Classification (Task 1–3)
2. Information Retrieval (Task 4)

For the first category, participants had to discover for a given recipe: the level of difficulty (4 levels), the type of dish (starter, main dish or desert) and the best title from a list of possible titles. For the ***I**nformation **R**etrieval (IR)*task, participants had to identify the ingredients for each recipe from a normalized list of possible ingredients. In our research we developed

[8]http://computercookingcontest.net

Fig. 5.11. General Description of the IR task (Task 4) form *DEFT 2013*



several systems for the IR task (Task 4) because of the possibility of using our syntactic features (*dependency sub–tree*). The description of the task is illustrated in Figure 5.11.

It is important to note that the normalized list of possible ingredients (i.e. ingredient pool), contains at least all the ingredients to be found in the recipes of the corpus. However, there is the possibility that the actual ingredient name is not explicitly in the recipe (e.g. "Olive Oil" in the recipe may be listed as just "Oil" or even "Fry the eggs", which implies the presence of "Oil"). Moreover, there might also be the ingredients that are not even present in the training corpus (Dini et al., 2013). This issues have been observed to effect the developed systems both by the participants on the original challenge and ours especially when lexical features were used. We also notice that in terms of algorithms the original contestants used *rule based*, *ML* and hybrid systems that have rule based components as well as ML components. Most participants including the best performing team used ML to classify ingredients on the basis of different features extracted form the recipes. Next, we shall present the evaluation metric used in the challenge and the results of the original participants.

Evaluation and Results of The Challenge

We used the evaluation platform designed specifically for the challenge, to evaluate our system's performance. The evaluation metric used for the task was ***M**ean **A**verage **P**recision (MAP)*. Let us consider that there are N recipes and for any recipe R_i there are n_i ingredients $\{I_i^1 \ldots I_i^j \ldots I_i^{n_i}\}$ and P be the precision, then the MAP metric is,

$$MAP = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{j=1}^{n_i}P\left(I_i^j\right)$$

There were 6 teams which participated in the challenge, of which 2 industrial participants and the rest from the academic arena. *Celi France* performed the best with a MAP of **0.6622**.

They presented a system that uses a hybrid approach to solve the given problem (Dini et al., 2013). They used a rule-based system to identify the potential ingredient and then filter them using a classifier on the basis of the type of the recipe (from task 2) and their system relied strongly on the lexical features (mostly lemma). The final MAP score of the top 5 teams are listed in table 5.9.

Table 5.9. DEFT Challenge Result (MAP Score)

Team	LIM&Bio	GREYC	LIA	Celi Fr.	Wikimeta
Run #1	0.4115	0.4881	*0.6287*	**0.6662**	0.5675
Run #2	0.4170	0.5074	0.6218	—	*0.6428*
Run #3	0.4649	0.5556	0.6191	—	—
Rank	5	4	3	1	2

Experiment Setup and Results

For our systems, the recipes were parsed using the same statistical dependency parser for French presented in § 4.4.2. We used the *Berkeley* parser for the preprocessing that establishes the dependencies between tokens. Among all the relations only *coordination* required some normalization. In the parser output the coordination dependencies are represented as a chain rather than a single relation (to maintain bi–lexical relations), i.e. N coordinated tokens are represented using a combination of two dependencies, `COORD`, connects the first conjunct with the first coordinator and `DEP-COORD` connect the next coordinator. Furthermore, multiple conjuncts are connected as a chain using the `COORD` with each other and the first element is connected to the *head* by a `DEP-COORD` relation. We resolved the `COORD` to a flat representation for some of the experiments to observe the effect on the performance. All the parsers output was in the same adapted *CoNLL*[9] data format described in § 4.4.2.

We automatically extracted the patterns using a script from the parsed output [10] then used them in our systems as features along with other features. We experimented primarily with two machine learning methods: *Logistic Regression* (Cramer, 2002) and *Perceptron* (Minsky and Papert, 1969). While training a system with the patterns, we mapped all the patterns extracted from each document to all the ingredients associated with that document. Formally, in the training set if we have n documents $D = \{d_1, d_2 \cdots d_n\}$ and m ingredients

[9] http://nextens.uvt.nl/depparse-wiki/DataFormat

[10] script available at https://github.com/eldams/ConLL-SimpleReader

$I = \{i_1, i_2 \cdots i_m\}$ then we can have the ingredient i^x given, $1 < x < m$ in $D^{'} \subset D$. All the features for the machine learning methods were calculated from this subset $D^{'}$. The detailed performance measured for our experimental systems are listed in Table 5.10.

Table 5.10. Experimental Results: Ingredient Extraction from Recipes

System & Features	MAP	P(5)	P(10)	P(100)	R(5)	R(10)	R(100)
`identify-tokens`	0.3564	0.4960	0.3556	0.0375	0.3607	0.4930	0.5114
`identify-lemmas`	0.4355	0.5402	0.4193	0.0456	0.4000	0.5893	0.6262
`identify-tokens+lemmas`	0.4430	0.5375	0.4249	0.0469	0.3990	0.5986	0.6428
`learn-lemmas`	0.7196	0.7409	0.5306	0.0695	0.5420	0.7446	0.9493
`learn-lemmas+mine`	0.7362	0.7565	0.5432	0.0695	0.5538	0.7615	0.9487
`learn-lemmas+mine+coord`	0.7364	0.7555	0.5431	0.0695	0.5532	0.7610	0.9490
`learn-lemma+pos`	0.7182	0.7414	0.5305	0.0695	0.5423	0.7446	0.9495
`learn-percept`	**0.7500**	0.7588	0.5547	0.0706	0.5545	0.7779	0.9648

In Table 5.10 all the systems with the prefix `identify` used string matching and represent the baselines for the task and all the system with the prefix `learn` are ML systems. Except for the `learn-percept` all the ML systems used *logistic regression* as the learning algorithm. The suffix of a system name states the features used (e.g. tokens, lemma etc.). the suffix element `mine` implies that the `Dependency Sub-Tree` patterns have been used as features. Although it is not explicitly mentioned, the *Perceptron* based system uses the patterns as features as well. The suffix element `coord` refers to the fact that the data has been normalized, following the discussion at the beginning of this sub–section and then used in the extraction process. The systems predicted a score for each ingredient associated with the pattern set extracted for a given recipe during the testing phase thus, outputs a ranked list of ingredients for the recipe. The columns titled $P(x)$ represent precision for a prediction when the first x elements are selected and the columns titled $R(x)$ are the equivalent recall values. The evaluation metric (i.e. *MAP*) however, puts more emphasis on precision and the comparison of our systems were performed based on the *MAP* score. One obvious observation is that as the number of top ingredients (x) increases the recall increases along and the precision decreases.

During performance analysis we found that for example, if `fry` is mentioned in a recipe that implicitly considers `oil` as an ingredient. Then if `oil` never appears in the recipe, there shall be no lexical map between `oil` and `fry`. But if *Dependency Sub–Tree* patterns are used for the training, any pattern that appearers in a recipe would potentially be linked to all the ingredient e.g. the verb `fry` will be linked to `oil`, regardless of its actual presence in the

recipe. It can be observed clearly that even the *logistic regression* produces higher scores when patterns are used in conjunction with lexical features. Although the performance of the *Perceptron* algorithm was the best, the first significant improvement can be already observed when feature to ingredient was mapped for each recipe (e.g. `learn-lemmas` shows about 30% improvement over the baseline). The use of the syntactic patterns improves the *MAP* between 2% and 4%. All the results using document level mapping showed better performance than the top system from the original challenge. We have used the syntactic features used in this experiment for the *Semi–Supervised* learning experiments that is presented next.

5.4.3 Semi–Supervised Machine Learning

Learning algorithms fall under the umbrella term of ***M**achine **L**earning (ML)*, are a great way to deal with problems where the mapping between the input and the output is unknown and complex. These methods are data driven and have a strong tie to the field of *Statistics*, especially *Computational Statistics*. Given the circumstances pertaining our research, i.e. lack of evaluation data and the subjective nature of the problem itself, it seemed reasonable to investigate the possibility of using ML methods in the problem domain. The nature of the data i.e. lack of large amount of annotated data leaves us with the option of *Semi–Supervised Learning* using *Unsupervised Clustering*. In this section we shall introduce the general field of ML, *Semi–Supervised learning* and *Unsupervised clustering*. We shall also present the experimental setup and results obtained from the *Semi–Supervised Learning* experiments.

Machine Learning

ML is an umbrella term used to represent the field in *Computer Science* that encompasses several somewhat interrelated fields: the most prominent of them are *Pattern Recognition* and *Computational Learning Theory*. The primary objective of ML is to develop algorithms and methods capable of learning the underlying model for a given set of data and later use the model to predict the nature of previously unseen data. ML, especially theory of learning shows strong similarity (even overlaps) with computational statistics and many of the algorithms have strong mathematical basis. A detail overview of the research questions pertaining to ML can be found in (Mitchell, 2006). Let us Consider a set of input $X = \{x_1, x_2, x_3 \ldots x_n\}$ and a set of output $Y = \{y_1, y_2, y_3 \ldots y_m\}$, thus, the broader goal of ML is to map, $f : X \rightarrow Y$, where, $y = f(x)$. There are some learning paradigm that evolved mostly to tackle specific types of problems, e.g. *Classification*, *Clustering* and *Regression*.

Classification methods are designed to map the inputs into a discrete output space, i.e. each input can be mapped into a set of discrete set of possible outputs. A special case of *classification* is the *Binary Classification* where the output space has only two elements. On the other hand, *Clustering*, is a spacial case of classification where not only the mapping function is unknown but also the output map is unknown, i.e. the number of classes is not known. Finally, *Regression* methods tries to map the input into a continuous output space instead of a discrete output space.

Furthermore, there are several learning setups used in conventional ML practice. *Supervised Learning* is most probably the most widely used among these setups. In supervised learning the learning algorithm attempts to learns the underlying map between the input and the output from a set of labelled example i.e. data points with the associated class or value. In contrast, *Unsupervised Learning* attempts to learn the underlying relation between the input and output from unlabelled examples, i.e. only the data points, thus, not only the algorithm has to find the mapping but also to group them to satisfy the problems original goal of producing a specific number of classes from the mapping. Unsupervised classification is often called *Clustering*. *Semi-Supervised Learning* is somewhat a middle ground, where a small number of labelled example with a lot of unlabelled example is used to identify the overall distribution features in a data to classify or estimate the value for the unlabelled examples. There are other setups e.g. *Reinforced Learning* that are less relevant to our research. We have used *Semi–Supervised Learning* using *Unsupervised Clustering* for our experiments due to empirical restrictions (lack of labelled data) and we shall provide further details in the next section.

Unsupervised Clustering

Unsupervised Clustering is a widely used method for classification problems with low supervised resources, i.e. where very little supervised data is available but some or large amount of unsupervised data is available. The objective is to identify hidden distributions in the data that can correlate to the target solution domain without any prior knowledge of the solution domain and having any reward or punishment system associated to the learning process. Unsupervised clustering is closely related to the problem of *Density Estimation* in statistics (Jordan and Bishop, 2014), which expresses the concept of the construction of an estimate, based on observed distribution of data, of an unobservable underlying probability density function. The most commonly used methods include, *K–Means*, *Mixture Models* and

Latent Variable Models (which interestingly uses blind signal separation techniques such as *PCA* and *SVD*). However, we have used *K–Means* clustering extensively for our experiments.

K-Means clustering is a method of vector quantization, originated from *Signal Processing*, that is popular for cluster analysis. *K-Means* clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Given a set of observations $X = \{x_1,\ x_2\ \ldots\ x_n\}$, where each observation is a *d-dimensional* real vector, *K-Means* algorithm attempts to partition them into $k\ (\leq\ n)$ sets $S = \{s_1,\ s_2\ \ldots\ s_k\}$ by identifying the means of each cluster. The cluster means also known as centroids are represented as a set of k elements: $\{\mu_1,\ \mu_2\ \ldots\ \mu_k\}$ (one for each cluster). The objective of the algorithm can be seen as,

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j\ -\ \mu_i||^2$$

Although the principal of *K-Means* has strong ties to the work of Steinhaus (1956), the original idea was proposed by *Stuart Lloyd* and as early as 1957 as a technique for *Pulse-Code Modulation*, although it was never published outside of *Bell Labs* until 1982 (Lloyd, 1982). The most common algorithm that uses an iterative refinement technique is called the *Lloyd's Algorithm*. A general form of the algorithm is presented in Appendix A § A.3. The algorithm has two distinct steps: the first is the assignment stage where each observation is assigned to the cluster whose mean yields the least ***W**ithin-**C**luster **S**um of **S**quares (WCSS)*. Since the sum of squares uses the squared *Euclidean Distance* and the *square root* is a monotone function, this also is the minimum *Euclidean Distance* assignment i.e. the "*nearest*" mean. It implies partitioning the observations according to the *Voronoi diagram* (see the article by Aurenhammer (1991) for details) generated by the means. The algorithm has converged when the assignments no longer change and both steps optimize the *WCSS* objective. Since there only exists a finite number of such partitioning, the algorithm must converge at least to a local optimum. However, there is no guarantee that the global optimum will be found using this algorithm. The other step is the update step where, the algorithm calculates the new means to be the centroids of the observations in the new clusters.

The complexity of the algorithm in a general Euclidean space d even for 2 clusters has been reported to be *NP-Hard* (Aloise et al., 2009). If clusters (k) and the dimension (d) are fixed, the problem can be solved in time $O(n^{dk+1} \log n)$, where n is the number of entities to be clustered (Inaba et al., 1994). The primary criticism of the method is the random generation and selection of centroids i.e. very little control over the original cluster

generation. Furthermore, the tendency to converge to a local optimum often leads to wrong clustering. However the iterative implementation with multiple centroid selection allows to overcome the first issue, but the fact remains that *K-Means* is an optimal method for a well clustered data. Selection of k can make significant difference in the outcome and is has been suggested to observe the behaviour of the data after dimension reduction to identify optimal value for k (Jardino, 2004). We shall demonstrate our experience with this clustering next.

Semi–Supervised Leaning Using K–Means

For the task of FP identification we chose the *K–Means* algorithm to develop our *Semi–Supervised* learning method. We used our small annotated reference data to evaluate the accuracy of the assigned labels of the clusters (as *FP Positive* or *FP Negative*) generated by the *K–Means* clustering from the unlabelled training dataset thus, adopting the notion of *Semi–Supervised* learning. For this experiment we used the *K–Means* algorithm to generate high number of clusters (odd numbers of clusters between 7 and 15) to identify the separations in the training data. The research objective is to separate the data into two groups, thus we generated all possible combination of two sets for the clusters form the original clusters and assigned the labels (*FP+* and *FP-*) systemically. Each pair represents a possible solution space and each solution space is evaluated for the accuracy in terms of predicting the labels for the labelled data. For example, if we have three clusters $\{1,\ 2,\ 3\}$, the possible sets are presented in Table 5.11,

Table 5.11. Cluster Set Distribution for Evaluation

Solution Space	FP Positive	FP Negative
1	[1]	[2, 3]
2	[1, 2]	[3]
3	[1, 3]	[2]
4	[2, 3]	[1]
5	[3]	[1, 2]
6	[2]	[1, 3]

Among the solution spaces produced the space with the best average *F2–Score* was considered to be the expected solution space for the feature space and for the given cluster size. We also used *SVD* to reduce the feature space for both the training and evaluation data to their respective lower dimensions (we applied the clustering algorithm for the dimensions 1 through 5). Thus for each feature space the number of solution space (S_s) is based on two

parameters, the cluster size (k) being used and the number of principal components (d) being used, i.e. $(k \times d)$. Thus if we are using n different cluster sizes $K = \{k_1, k_2, k_3, \ldots k_n\}$ and m principal components, $D = \{1, 2, 3, \ldots m\}$, for the feature space f^x then the total number solution spaces shall be,

$$S_s^x = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} k_i \times j$$

Eventually, the training performance on a feature space is the best performing solution space based on the aforementioned criterion (best average *F2–Score*). We however gave preference to a solution space with lower number of clusters and dimensions, in case of two solution spaces have the same average *F2–Score* because we decided to reward any possible reduction in terms of processing complexity. There were two other parameters for the *K–Means* algorithm that we experimented with beforehand and choose the values based on a reasonable balance between processing complexity and finding the best performing the model for the reference data. The two parameters to be adjusted were: number of maximum iteration the algorithm will perform to archive convergence and the number of times the centroid should be assigned to a random data point. We set the value of iteration to 1500 and the centroid assignment to 300 and found that in all the feature spaces it was sufficient to achieve the maximum possible performance. However, we are aware of the fact that *K–Means* algorithm is notorious to converge on local maxima[11] instead of continue iterating to achieve global maxima. Thus, the algorithm can fail to achieve the optimal solution for a given scenario and so, we also fitted our experiments with another level of iteration to run each training scenario I times. For our experiments we used $I = 15$ and again we found it adequate to achieve possible best performance.

Table 5.12. Semi–Supervised Training Performance: Best

Annotator	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

The *Semi–Supervised* method was applied to all the feature spaces and the performance of the best solution space was recorded for each of them. We also stores the feature space

[11] in reality the learning objective is to find the global minima rather than maxima in terms of minimum squared distance form the centroid, although it is a matter of semantics and the perception of the learning goals and not some rules carved in stones

definition, solution space definition, i.e. the label for each cluster and the model for future use. The best performances that fitted our research objectives are listed in tables below. The presented results are the training performance and we have relative confidence based on the available evaluation data. Furthermore, we acknowledge the possibility of over–fitting the reference data in these models. However, this performance is achieved based on the

Table 5.13. Experimental Parameters: Best Performance

Feature Space	Principal Components (*PC*)	Cluster Number (*k*)
`surface_extended_dgram`	3	17
`lemma_gpos_unigram`	4	15
`gpos_trigram`	3	13
`gpos_dgram_wild`	5	15

underlying distribution of the data points for a given features set and most definitely the bias imposed by the lexical resources have been lifted. The best performance was based on the achieved best average *F2-Score* for a given feature space bust also based on extracting maximum number of positive pages. The aforementioned results (Table 5.12) were obtained for the setups listed in 5.13.

One obvious observation from these results were that both *lexical* and *syntactic* features are present and there is no significant pattern regarding *PC* or cluster numbers. Furthermore, there was one *false negative* and as we discussed earlier, this page was annotated by only one annotator (*BDF_MG*). The results are the possible best clustering for the given reference data and without additional evaluation we have to assume possible over–fitting nature of the models. However, we can also conclude that there exist a distribution that fits the reference data and with more evaluation data we can identify the generalized form of this distribution. The other hypothesis was the performance of the syntactic features and the best performance of the *Dependency Sub–Tree* feature space is presented below.

Table 5.14. Semi–Supervised Training Performance: `surface_dep_subtree`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	15	10	2	15	0	1	0.833	0.909	0.962	0.882	1	0.926
BdF_DB	5	15	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778
BdF_MG	5	15	11	1	15	0	1	0.917	0.957	0.982	0.938	1	0.963
PSA	5	15	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778

Given the nature of the reference data set, Table 5.14 is the best possible clustering performance if we want to maximize *true positives* and minimize *false positives*. However, the same performance has been observed for `spos_quardgram`, which is a morphological feature space. Furthermore, in case of `spos_quardgram` the performance can be achieved with $PC = 3$ and $k = 13$, thus, performed a little better than the *Sub–Tree* feature space. One notable aspect of all the clustering evaluation data is that the performance is based on 27 pages rather than the original 37 pages. The 10 pages were ignored because of lack of significant number of features thus reinforcing one of our primary objectives of reducing the number of pages to read.

Chapter 6

Final Thoughts & Future Prospects

This research has produced some interesting results but also opened up the opportunity to ask a whole range of new questions. We consider this line of investigation towards total automation of ***F**unction **P**oint (FP)* identification far from being concluded. This research will only server as a launch pad for further questions and prospects of future research. In this work, we focused on adopting a practical approach towards identifying possible solutions for a real world problem. Lack of data, especially annotated data made the task even more difficult, however, we believe that we have established a platform that will allow incorporating new data with minimum amount of effort. Thus, further research can be possible and benefit from our findings. Our contributions can be listed as follows,

- We have successfully implemented a complete data processing chain to generate corpus from software specification document produced at different stages of the software development life cycle. The chain can take *PDF* format files and produce corpus with multiple layers of annotation including *FP* annotation. Currently, the processing chain is adapted for *French*, however multi–language support can be implemented with minimum effort.
- We have generated a corpus containing approximately 41,000 sentences exclusively for FP research. To the best of our knowledge, no such resource existed so far. The corpus contains 11,250 unique lemmas with detailed morphological annotation and dependency relations, that can be valuable resource for future linguistic and computational analysis.
- We have demonstrated a fine grained inter–annotator agreement analysis for FP identification task. We have presented the annotation guidelines and implemented the evaluation platform dedicated for the task. The adoption of multi–level granularity for the evaluation provides more information regarding the annotation quality than the standard used by most researchers in FP research i.e. using final FP count for agreement analysis.

- In this thesis we have also presented our analysis of the data and reported observed patterns. The correlation between different features and FP has been observed in the data. Although due to the lack of annotated data it is rather difficult to assign high confidence to these relations. Only farther testing can put the results into perspective.
- Our analysis also provided valuable information about some of the features that showed stronger correlation to the existence of FP than the others. We would like to mention our observation that syntactic dependency seem to show stronger correlation than the lexical features.
- We presented a text representation model (i.e. dependency sub–tree) that has been used successfully to achieve significant performance gain in the cooking recipes domain where large amount of supervised data was available (see § 5.2.2). We have also demonstrated the correlation observed in the FP domain for the same model thus, a significant feature has been identified.
- We have also presented the significance of first principal component of the feature space and the use of multi–cluster combinations to achieve better clustering performance. The clustering results at least demonstrated good training performance for syntactic patterns over lexical features.

The contributions are not disjoint from future prospects of research. We have assessed several research possibilities that could not be investigated due to various shortcomings. We would like to address those issues and ideas in the next section,

- Introducing large amount of data especially, annotated data would significantly change the overall aspect of the research in terms of detailed data analysis and methods that can be employed for the task. With more data we can test our current method and establish the level of confidence of our model. With more annotation the level subjectivity that currently exists in the evaluation process will be reduced.
- With large amount of annotated data we can experiment with supervised and semi–supervised learning algorithms in search of better performing models. Semi–supervised methods can even be used with less annotated data, thus more data whether annotated or not should improve the current state of the research.
- We have not used any semantic feature or resources so far, however we think it should help us identifying FPs. Among the frameworks, some level of ***W**ord **S**ense **D**isambiguation (WSD)* is necessary and resources such as WordNet (Miller, 1995) can be used for these types of lexical semantic based research. There is a similar resource for *French* called *WOLF* (Sagot and Fišer, 2008a,b) freely available for use.
- For the modelling of detailed and specific function points we also need to explore semantic possibilities at higher level textual groups e.g. sentences. Making semantic models for FPs would be the ultimate objective of this research. We have looked into *FrameNet* (Baker et al., 1998) for the semantic modelling of FP by correlating frames associated with potential lexical entities (we are mostly interested in verbs). However, *FrameNet*

development for *French* was under development at the time of writing this thesis and it is far from a release. Thus, we would like to propose manual identification of frames from the *English FrameNet* and use them after automatic translation. This is not only a rather tedious process but also each resultant frame is needed to be evaluated by professional FP analysts in a objective manner before we can use them.

- This research can potentially be adopted for the multi–lingual FP analysis. We recognize the lack of data in other languages, yet at least for some resource rich languages, it is a possibility.

References

Abeillé, A. and Barrier, N. (2004). Enriching a French Treebank. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC)*.

Alberts, I. and Forest, D. (2012). Email Pragmatics and Automatic Classification: A Study in the Organizational Context. *Journal of the American Society for Information Science and Technology*, 63(5):904–922.

Albrecht, A. J. (1979). Measuring Application Development Productivity. In *SHARE/GUIDE IBM Application Development Symposium*, pages 83–92, Monterey, CA, USA.

Albrecht, A. J. and Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, SE-9(6):639–648.

Alias-i (2008). LingPipe 4.1.0. http://alias-i.com/lingpipe. accessed: 20 December, 2015.

Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). NP-Hardness of Euclidean Sum-of-Squares Clustering. *Machine Learning*, 75(2):245–248.

Andersen, P. M., Hayes, P. J., Huettner, A. K., Schmandt, L. M., Nirenburg, I. B., and Weinstein, S. P. (1992). Automatic extraction of facts from press releases to generate news stories. In *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC '92, pages 170–177, Stroudsburg, PA, USA. Association for Computational Linguistics.

Appelt, D. E. (1999). Introduction to information extraction. *AI Commun.*, 12(3):161–172.

April, A., Merlo, E., and Abran, A. (1997). A reverse engineering approach to evaluate function point rules. In *Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on*, pages 236–245.

Armano, G. (2015). Modelling progressive filtering. *Fundamenta Informaticae*, 138(3):285–320.

Asadullah, M., Nouvel, D., and Paroubek, P. (2014a). Using verb–noun patterns to detect process inputs. In Sojka, P., Horák, A., Kopeček, I., and Pala, K., editors, *Text, Speech and Dialogue*, volume 8655 of *Lecture Notes in Computer Science*, pages 181–188. Springer International Publishing.

Asadullah, M., Paroubek Patrick, and Vilnat, A. (2014b). Bidirectionnal Converter Between Syntactic Annotations: from French Treebank Dependencies to PASSAGE Annotations, and Back. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, LREC-2014, pages 2342–2347. European Language Resources Association (ELRA).

Aurenhammer, F. (1991). Voronoi Diagrams–A Survey of A Fundamental Geometric Data Structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.

Bach, N. and Badaskar, S. (2007). A review of relation extraction. *Language Technologies Institute, Carnegie Mellon University*.

Bajaj, K. S. and Pieprzyk, J. (2014). A Case Study of User–level Spam Filtering. In *Proceedings of the Twelfth Australasian Information Security Conference - Volume 149*, AISC '14, pages 67–75, Darlinghurst, Australia. Australian Computer Society, Inc.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet Project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, pages 86–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

Banker, R. D., Kauffman, R. J., Wright, C., and Zweig, D. (1994). Automating output size and reuse metrics in a repository-based computer-aided software engineering (case) environment. *Software Engineering, IEEE Transactions on*, 20(3):169–187.

Banko, M. and Etzioni, O. (2008). The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, ACL 2008, pages 28–36, Columbus, Ohio. Association for Computational Linguistics.

Batista, V. A., Peixoto, D. C. C., Borges, E. P., Pádua, W., Resende, R. F., and Pádua, C. I. P. S. (2011). ReMoFP: A Tool for Counting Function Points from UML Requirement Models. *Advances in Software Engineerin*, 2011:1:1–1:7.

Bjørner, S. and Ardito, S. C. (2003). Online Before the Internet, Part 1: Early Pioneers Tell Their Stories. *Searcher: The Magazine for Database Professionals*, 11(6).

Bloomfield, L. (1933). *Language*. Holt, New York.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA. ACM.

Bobrow, D. G. (1964). A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I*, AFIPS '64 (Fall, part I), pages 591–614, New York, NY, USA. ACM.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, USA.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., and Horowitz, E. (2000). *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Bonissone, P. P. and Dutta, S. (1990). Mars: A mergers and acquisitions reasoning system. *Computer Science in Economics and Management*, 3(3):239–268.

Brin, S. and Page, L. (1998). The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

Brown, D. (1990). Automated Function point Counting - Myth or Reality. In *Proceedings of the IFPUG Fall Conference*, pages 168–180.

Brownson, H. L. (1960). Research on Handling Scientific Information. *Science*, 132:1922–1931.

Buchholz, S. and Marsi, E. (2006). CoNLL–X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL–X '06, pages 149–164, Stroudsburg, PA, USA. Association for Computational Linguistics.

Buglione, L. (2008). *Misurare il software. Quantità, qualità, standard e miglioramento di processo nell'Information Technology*. Informatica e organizzazioni. Franco Angeli.

Bundschuh, M. and Dekkers, C. (2008). *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement*. Springer Publishing Company, Incorporated, 1 edition.

Bush, V. (1945). As We May Think. *Atlantic Monthly*, 176(1):641–649.

Candito, M., Crabbé, B., and Denis, P. (2010). Statistical French Dependency Parsing: Treebank Conversion and First Results. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*, LREC '10, Valletta, Malta. European Language Resources Association (ELRA).

Candito, M., Crabbé, B., Denis, P., and Guérin, F. (2009). Analyse syntaxique du français : des constituants aux dépendances. In *16e Conférence sur le Traitement Automatique des Langues Naturelles - TALN 2009*, Senlis, France.

Candito, M., Nivre, J., Denis, P., and Anguiano, E. H. (2010). Benchmarking of Statistical Dependency Parsers for French. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 108–116, Stroudsburg, PA, USA. Association for Computational Linguistics.

Carbonell, J. and Goldstein, J. (1998). The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 335–336, New York, NY, USA. ACM.

Carlson, L., Onyshkevych, B., and Okurowski, M. E. (1993). Corpora and data preparation. In *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*.

Carroll, G. and Charniak, E. (1992). Two Experiments on Learning Probabilistic Dependency Grammars from Corpora. Technical Report CS-92-16, Department of Computer Science, Brown University, Providence, RI.

Carroll, J. A. (2000). Statistical Parsing. In *Handbook of Natural Language Processing*, pages 525–544. Dekker, New York.

C&EN (1954). New tools for the resurrection of knowledge. *Chemical & Engineering News Archive*, 32(9):866–869.

Ceri, S., Fraternali, P., and Bongio, A. (2000). Web modeling language (webml): A modeling language for designing web sites. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 33(1–6):137–157.

Chao, H. and Fan, J. (2004). Layout and Content Extraction for PDF Documents. In Marinai, S. and Dengel, A., editors, *Document Analysis Systems*, Lecture Notes in Computer Science, pages 213–224. Springer.

Choi, S., Park, S., and Sugumaran, V. (2006). Function Point Extraction Method from Goal and Scenario Based Requirements Text. In *Natural Language Processing and Information Systems*, volume 3999 of *Lecture Notes in Computer Science*, pages 12–24. Springer Berlin Heidelberg.

Choi, S., Park, S., and Sugumaran, V. (2012). A rule-based approach for estimating software development cost using function point and goal and scenario based requirements. *Expert Systems with Applications: An International Journal*, 39(1):406–418.

Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1968). *Remarks on Nominalization*. Linguistics Club, Indiana University.

Ciaramita, M. and Altun, Y. (2005). Named-entity recognition in novel domains with external lexical knowledge. *Proceedings of the NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*.

Clark, J. H. and González Brenes, J. P. (2008). Coreference resolution: Current trends and future directions. *Language and Statistics II Literature Review*, pages 1–14.

Cleverdon, C. W. (1959). The Evaluation of Systems Used in Information Retrieval. In *Proceedings of the International Conference on Scientific Information Washington, D.C., Nov. 16-21, 1958 – Volume 1*, pages 687–698. National Academy of Sciences, National Research Council.

Cleverdon, C. W. (1991). The Significance of the Cranfield Tests on Index Languages. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '91, pages 3–12, New York, NY, USA. ACM.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Cohen, W. W. (1996). Learning Trees and Rules with Set-Valued Features. In Clancey, W. J. and Weld, D. S., editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, August 4-8, 1996, Volume 1.*, AAAI '96, IAAI '96, pages 709–716. AAAI Press / The MIT Press.

Cooper, W. S., Gey, F. C., and Dabney, D. P. (1992). Probabilistic Retrieval Based on Staged Logistic Regression. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '92, pages 198–210, New York, NY, USA. ACM.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley–Interscience, New York, NY, USA.

Covington, M. A. (1984). *Syntactic Theory in The High Middle Ages*. Cambridge University Press.

Cowie, J. and Lehnert, W. (1996). Information extraction. *Commun. ACM*, 39(1):80–91.

Cowie, J. R. (1983). Automatic analysis of descriptive texts. In *Proceedings of the First Conference on Applied Natural Language Processing*, ANLC '83, pages 117–123, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cramer, J. (2002). The Origins of Logistic Regression. Tinbergen Institute Discussion Papers 02-119/4, Tinbergen Institute.

Cucerzan, S. and Brill, E. (2004). Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 293–300, Barcelona, Spain. Association for Computational Linguistics.

Cullingford, R. E. (1978). *Script Application: Computer Understanding of Newspaper Stories*. PhD thesis, Department of Computer Science, Yale University, New Haven, CT.

Cunningham, H., Gaizauskas, R., and Wilks, Y. (1995). A General Architecture for Text Engineering (GATE) – a new approach to Language Engineering R&D. Technical Report CS – 95 – 21, Department of Computer Science, University of Sheffield.

Daumé III, H. and Marcu, D. (2005). A large-scale exploration of effective global features for a joint entity detection and tracking model. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 97–104, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Davis, J. and Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA. ACM.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science (JASIS)*, 41(6):391–407.

DeJong, G. (1979). Prediction and substantiation: A new approach to natural language processing. *Cognitive Science*, 3(3):251–273.

DeJong, G. (1982). An Overview of the FRUMP System. In *Strategies for Natural Language Processing*, pages 149–176. Lawrence Erlbaum, Hillsdale, NJ.

del Bianco, V., Gentile, C., and Lavazza, L. (2008). An evaluation of function point counting based on measurement-oriented models. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 11–20, Swinton, UK. British Computer Society.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Denis, P. and Baldridge, J. (2007). Joint determination of anaphoricity and coreference resolution using integer programming. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 236–243, Rochester, New York. Association for Computational Linguistics.

Dennis, B. K., Brady, J. J., and Dovel Jr., A. (1962). Index Manipulation and Abstract Retrieval by Computer. *Journal of Chemical Documentation*, 2(4):234–242.

Dewey, M. (1876). *A Classification and Subject Index, for Cataloguing and Arranging the Books and Pamphlets of a Library*. Cass, Lockwood & Brainard Company.

Dini, L., Bittar, A., and Ruhlmann, M. (2013). Approches hybrides pour l'analyse de recettes de cuisine DEFT, TALN-RECITAL 2013. In *Actes de DEFT 2013 : 9e DÉfi Fouille de Textes*, pages 53–65, Les Sables d'Olonne, France.

Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., and Weischedel, R. (2004). The automatic content extraction (ace) program, tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC-2004)*, Lisbon, Portugal. European Language Resources Association (ELRA).

Doyle, L. B. and Becker, J. (1975). *Information Retrieval and Processing*. Melville Publishing Co., Los Angeles, USA.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley–Interscience.

Dumais, S., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., and Robbins, D. C. (2003). Stuff I've Seen: A System for Personal Information Retrieval and Re-use. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 72–79, New York, NY, USA. ACM.

Dumke, R. and Abran, A. (2011). *COSMIC Function Points: Theory and Advanced Practices*. CRC Press.

Edward, S. H. (1920). Means for Compiling Tabular and Statistical Data. US Patent 1,351,692.

Eliot, S. and Rose, J. (2009). *A Companion to the History of the Book*. Blackwell Companions to Literature and Culture. John Wiley & Sons.

Ellafi, R. and Meli, R. (2006). A Source Code Analysis-Based Function Point Estimation Method integrated with a Logic Driven Estimation Method. In *Proceedings of the Software Measurement European Forum*, SMEF '06, pages 177–194.

Elshoff, J. L. (1978). An investigation into the effects of the counting method used on software science measurements. *SIGPLAN Notices*, 13(2):30–45.

Evert, S. (2004). Significance Tests for The Evaluation of Ranking Methods. In *Proceedings of Coling 2004*, Geneva, Switzerland.

Faure, D. and Nédellec, C. (1999). Knowledge Acquisition of Predicate Argument Structures from Technical Texts Using Machine Learning: The System ASIUM. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management*, EKAW '99, pages 329–334, London, UK. Springer–Verlag.

Fenichel, R. (1979). Surveyor's forum: Heads i win, tails you lose. *ACM Computing Surveys (CSUR)*, 11(3):277.

Fenton, N. and Pfleeger, S. L. (1997). *Software Metrics (2Nd Ed.): A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition.

Fetcke, T. (1999). The Warehouse Software Portfolio, A Case Study in Functional Size Measurement. Technical Report 1999–20, Département d'informatique, Université du Québec à Montréal, Montréal, Canada.

Fitsos, G. P. (1979). Software science counting rules and tuning methodology. Technical report TR 03.075, IBM Santa Teresa Laboratory, San Jose, CA.

Fitzpatrick, J. (1997). Applying the abc metric to c, c++, and java. *C++ Report*.

Fraternali, P., Tisi, M., and Bongio, A. (2006). Automating Function Point Analysis with Model Driven Development. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06, Riverton, NJ, USA. IBM Corp.

Fuhr, N. (1989). Optimum Polynomial Retrieval Functions Based on the Probability Ranking Principle. *ACM Transactions on Information Systems (TOIS)*, 7(3):183–204.

Fuhr, N. and Buckley, C. (1991). A Probabilistic Learning Approach for Document Indexing. *ACM Transactions on Information Systems (TOIS)*, 9(3):223–248.

Gaifman, H. (1965). Dependency Systems and Phrase–Structure Systems. *Information and Control*, 8(3):304–337.

Gaizauskas, R. and Wilks, Y. (1997). Information extraction: Beyond document retrieval. Technical report CS – 97 – 10, Department of Computer Science, University of Sheffield.

Galton, A. and Mizoguchi, R. (2009). The Water Falls but the Waterfall Does Not Fall: New Perspectives on Objects, Processes and Events. *Applied Ontology*, 4(2):71–107.

Gencel, C. and Demirors, O. (2008). Functional Size Measurement Revisited. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 17(3):15:1–15:36.

Goldberg, E. (1931). Statistical Machine. US Patent 1,838,389.

Gopal, S. and Yang, Y. (2010). Multilabel Classification with Meta–level Features. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 315–322, New York, NY, USA. ACM.

Goutte, C. and Gaussier, É. (2005). A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In *ECIR*, volume 3408 of *Lecture Notes in Computer Science*, pages 345–359. Springer.

Great Britain Treasury, C. C. and Telecommunications Agency (2000). *SSADM Foundation*. Business Systems Development with SSADM. Stationery Office.

Grishman, R. and Sundheim, B. (1995). Design of The MUC-6 Evaluation. In *Proceedings of the 6^{th} Conference on Message Understanding*, MUC6 '95, pages 1–11, Stroudsburg, PA, USA. Association for Computational Linguistics.

Grishman, R. and Sundheim, B. (1996). Message understanding conference–6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 466–471, Stroudsburg, PA, USA. Association for Computational Linguistics.

Groß, T. M. (1994). *Theoretical Foundations of Dependency Syntax*. München: iudicium.

Grouin, C., Zweigenbaum, P., and Paroubek, P. (2013). DEFT2013 se met à table : présentation du défi et résultats. In *Actes de DEFT 2013 : 9e DÉfi Fouille de Textes*, pages 1–14, Les Sables d'Olonne, France.

Guthrie, D., Allison, B., Liu, W., Guthrie, L., and Wilks, Y. (2006). A Closer Look at Skip-gram Modelling. In *Proceedings of the Fifth international Conference on Language Resources and Evaluation*, LREC-2006, Genoa, Italy.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

Halliday, M. A. K. and Hasan, R. (1976). *Cohesion in English*. English Language Series. Longman, London.

Halstead, M. H. (1977). *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.

Harman, D. (2005). The History of IDF and Its Influences on IR and Other Fields. In *Charting a New Course: Natural Language Processing and Information Retrieval*, volume 16 of *The Kluwer International Series on Information Retrieval*, pages 69–79. Springer Netherlands.

Harris, Z. (1958). Linguistic Transformations for Information Retrieval. In *Proceedings of the International Conference on Scientific Information*, Washington, D.C. National Academy of Sciences – National Research Council.

Hassan, T. and Baumgartner, R. (2005). Intelligent Text Extraction from PDF Documents. In *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA 2005) and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2005)*, volume 2, pages 2–6, Vienna, Austria.

Hays, D. G. (1964). Dependency Theory: A Formalism and Some Observations. *Language*, 40(4):511–525.

Hellwig, P. (1986). Dependency Unification Grammar. In *Proceedings of the 11th International Conference on Computational Linguistics*, COLING '86, pages 195–198, Bonn, Germany. Institut für angewandte Kommunikations- und Sprachforschung e.V. (IKS).

Hellwig, P. (2003). Dependency Unification Grammar. In *Dependency and Valency*, volume Part 1 of *Handbooks of Linguistics and Communication Science*, pages 593–635. Walter de Gruyter.

Heycock, C. (2012). Specification, Equation, and Agreement in Copular Sentences. *The Canadian Journal of Linguistics / La Revue Canadienne De Linguistique*, 57:209–240.

Hiemstra, D. (1998). A Linguistically Motivated Probabilistic Model of Information Retrieval. In *Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL 1998*, volume 1513 of *Lecture Notes in Computer Science*, pages 569–584, Berlin, Germany. Springer Verlag.

Higgins, F. R. (1973). *The Pseudo–Cleft Construction in English*. PhD thesis, Massachusetts Institute of Technology (MIT), Cambridge, MA, United States.

Hirst, G. (1981). *Anaphora in Natural Language Understanding : A Survey*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, New York.

Ho, V. T. and Abran, A. (1999). A Framework for Automatic Function Point Counting from Source Code. In *International Workshop on Software Measurement (IWSM'99)*.

Hobbs, J. R. (1976). *Pronoun Resolution*. City College, Department of Computer Sciences.

Holmstrom, J. E. (1948). Section III. Opening Plenary Session. In *The Royal Society Scientific Information Conference*, Report and Papers Submitted. Royal Society, London, U.K.

Hudson, R. (1990). *English Word Grammar*. Blackwell, Oxford, UK.

Hull, D. (1993). Using Statistical Testing in the Evaluation of Retrieval Experiments. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 329–338, New York, NY, USA. ACM.

Huttunen, S., Yangarber, R., and Grishman, R. (2002). Complexity of event structure in ie scenarios. In *19th International Conference on Computational Linguistics (COLING)*, Taipei, Taiwan.

Inaba, M., Katoh, N., and Imai, H. (1994). Applications of Weighted Voronoi Diagrams and Randomization to Variance-based K-clustering: (Extended Abstract). In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, pages 332–339, New York, NY, USA. ACM.

Jaccard, P. (1901). Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272.

Jackendoff, R. (1977). *$\overline{X}$ Syntax: A Study of Phrase Structure*. MIT Press, Cambridge, MA, USA.

Jacobs, P. S. and Rau, L. F. (1990). Scisor: Extracting information from on-line news. *Commun. ACM*, 33(11):88–97.

Jahoda, G. (1961). Electronic Searching. In Shaw, R. R., editor, *The State of the Library Art*, pages 139–320. The Graduate School of Library Service, Rutgers–The State University, New Brunswick, New Jersey, USA.

James, M. (1985). *Classification Algorithms*. Wiley–Interscience.

Jardino, M. (2004). Recherche de structures latentes dans des partitions de « textes » de 2 à K classes . In *Le Actes des 7èmes Journées internationales d'Analyse statistique des Données Textuelles*, JADT–2004, pages 661–671. Presses Universitaires de Louvain (PUL).

Järvinen, T. and Tapanainen, P. (1998). Towards an Implementable Dependency Grammar. In *Proceedings of the ACL Workshop on Processing of Dependency-Based Grammars*, CoLing–ACL'98, pages 1–10.

Jebara, T. (2003). *Machine Learning: Discriminative and Generative (Kluwer International Series in Engineering and Computer Science)*. Kluwer Academic Publishers, Norwell, MA, USA.

Ji, H. and Grishman, R. (2011). Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 1148–1158, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ji, H., Westbrook, D., and Grishman, R. (2005). Using semantic relations to refine coreference decisions. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 17–24, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Joachims, T. (2002). Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA. ACM.

Jobson, J. D. and Korkie, B. M. (1980). Estimation for Markowitz Efficient Portfolios. *Journal of the American Statistical Association*, 75(371):544–554.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer Series in Statistics. Springer–Verlag, New York, NY.

Jones, C. (1978). Measuring programming quality and productivity. *IBM Systems Journal*, 17(1):39–63.

Jones, C. (1991). *Applied Software Measurement: Global Analysis of Productivity and Quality*. McGraw Hill Professional. McGraw-Hill, Education, New York, NY, USA.

Jones, C. (1996). Using Function Points to Evaluate CASE Tools, Volume 4. Technical report, Software Productivity Research Inc.

Jones, C. (2004). Software project management practices: Failure versus success. *CrossTalk: The Journal of Defense Software Engineering*, 17.

Jones, C. (2007). *Estimating Software Costs : Bringing Realism to Estimating: Bringing Realism to Estimating*. McGraw-Hill's AccessEngineering. Mcgraw-hill.

Jones, K. S. (1988). A Statistical Interpretation of Term Specificity and Its Application in Retrieval. In Willett, P., editor, *Document Retrieval Systems*, pages 132–142. Taylor Graham Publishing, London, UK.

Jordan, M. I. and Bishop, C. M. (2014). Neural Networks. In *Computing Handbook, Third Edition: Computer Science and Software Engineering*, pages 42: 1–24. CRC Press.

Kahane, S., Nasr, A., and Rambow, O. (1998). Pseudo–Projectivity, A Polynomially Parsable Non–Projective Dependency Grammar. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 646–652, Montreal, Quebec, Canada. Association for Computational Linguistics.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429.

Kemerer, C. F. (1993). Reliability of Function Points Measurement: A Field Experiment. *Communications of the ACM*, 36(2):85–97.

Kim, J., Park, S., and Sugumaran, V. (2006). Improving Use Case Driven Analysis Using Goal and Scenario Authoring: A Linguistics-based Approach. *Data & Knowledge Engineering*, 58(1):21–46.

Kitchenham, B. (1997). Counterpoint: The problem with function points. *Software, IEEE*, 14(2):29–31.

Kleinberg, J. M. (1999). Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM (JACM)*, 46(5):604–632.

Klusener, S. (2003). Source Code Based Function Point Analysis for Enhancement Projects. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 373–376.

Kusumoto, S., Edagawa, T., and Higo, Y. (2008). On an Automatic Function Point Measurement from Source Codes. In *2nd Workshop on Accountability and Traceability in Global Software Engineering (ATGSE2008)*, pages 27–28.

Kusumoto, S., Imagawa, M., Inoue, K., Morimoto, S., Matsusita, K., and Tsuda, M. (2002). Function Point Measurement from Java Programs. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 576–582, New York, NY, USA. ACM.

Lamma, E., Mello, P., and Riguzzi, F. (2004). A System for Measuring Function Points from an ER-DFD Specification. *The Computer Journal*, 47(3):358–372.

Lang, K. (1995). NewsWeeder: Learning to Filter Netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Lassez, J.-L., Knijff, v. d. D., Shepherd, J., and Lassez, C. (1981). A critical examination of software science. *Journal of Systems and Software*, 2(2):105 – 112.

Lay, David C., L., Steven R., and McDonald, J. J. (2015). *Linear Algebra and Its Applications (5th Edition)*. Pearson.

Léon, J., Memmi, D., Ornato, M., Pomian, J., and Zarri, G. P. (1982). Conversion of a french surface expression into its semantic representation according to the reseda metalanguage. In *Proceedings of the 9th Conference on Computational Linguistics - Volume 1*, COLING '82, pages 183–189, Praha, Czechoslovakia. Academia Praha.

Linsker, R. (1988). Self–organization in a perceptual network. *Computer*, 21(3):105–117.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Low, G. C. and Jeffery, D. R. (1990). Function points in the estimation and evaluation of the software process. *Software Engineering, IEEE Transactions on*, 16(1):64–71.

Luhn, H. P. (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1(4):309–317.

Luhn, H. P. (1958). The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2(2):159–165.

Luo, X. (2007). Coreference or not: A twin model for coreference resolution. In *HLT-NAACL*, pages 73–80. The Association for Computational Linguistics.

Lytinen, S. L. and Gershman, A. (1986). ATRANS: Automatic processing of money transfer messages. In *Proceedings of the 5th National Conference on Artificial Intelligence. Volume 2: Engineering*, AAAI-86, pages 1089–1093, Philadelphia, PA, USA. Morgan Kaufmann.

MacDonell, S. G. (1994). Comparative Review of Functional Complexity Assessment Methods for Effort Estimation. *Software Engineering Journal*, 9(3):107–116.

MacKay, R. J. and Oldford, R. W. (2000). Scientific method, statistical method and the speed of light. *Statistical Science*, 15(3):254–278.

Madi, A., Zein, O. K., and Kadry, S. (2013). On the improvement of cyclomatic complexity metric. *International Journal of Software Engineering & Its Applications*, 7(2).

Magerman, D. M. (1995). Statistical Decision-tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pages 276–283, Stroudsburg, PA, USA. Association for Computational Linguistics.

Malenge, J.-P. (1980). Critique de la physique du logiciel (critique of software science). Publication Informatique (Technical report) IMAN-P-23, Universite de Nice. France.

Mankiewicz, R. (2004). *The Story of Mathematics*. Mathematics (Princeton University Press). Princeton University Press.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Maron, M. E., Kuhns, J. L., and Ray, L. C. (1959). Probabilistic Indexing. a Statistical Technique for Document Identification and Retrieval. Technical Report 3, DTIC Document.

Matsuzaki, T., Miyao, Y., and Tsujii, J. (2005). Probabilistic CFG with Latent Annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 75–82, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mazzucco, F. A. (1990). Automation of Function Point Counting – An Update. In *Proceedings of the IFPUG Spring Conference*.

Mazzucco, F. A. (1992). IEF - Automatic Function Point Count. In *Proceedings of the 1992 IFPUG Conference*, pages 2–5, Baltimore, MA, USA.

McBryan, O. A. (1994). GENVL and WWWW: Tools for Taming The Web. In *Proceedings of the first World Wide Web Conference*, pages 79–90, Geneva, Switzerland.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320.

McCallum, A. K. (1996). Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering. http://www.cs.cmu.edu/~mccallum/bow. accessed: 20 December, 2015.

McCallum, A. K. (2002). MALLET: A Machine Learning for Language Toolkit. http://www.cs.umass.edu/~mccallum/mallet. accessed: 20 December, 2015.

McDonald, R. (2006). *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. PhD thesis, University of Pennsylvania.

McDonald, R., Pereira, F., Kulick, S., Winters, S., Jin, Y., and White, P. (2005). Simple algorithms for complex relation extraction with applications to biomedical ie. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 491–498, Stroudsburg, PA, USA. Association for Computational Linguistics.

Meister, D. and Sullivan, D. J. (1967). *Evaluation of User Reactions to a Prototype On–line Information Retrieval System*. CR. 918. Clearinghouse for Federal Scientific and Technical Information.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Mendes, O. (1997). Développement d'un protocole d'évaluation pour les outils informatisés de comptage automatique de points de fonction. Master's thesis, Université du Québec à Montréal.

Mendes, O., Abran, A., and Bourque, P. (1996). Function Point Tool Market Survey. Research report, Software Engineering Management Laboratory, Université du Québec à Montréal.

Metzler, D. and Croft, W. B. (2005). A Markov Random Field Model for Term Dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479, New York, NY, USA. ACM.

Middleton, W. E. K. (1966). *A History of The Thermometer And Its Use in Meteorology*. Johns Hopkins Press.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.

Minsky, M. and Papert, S. (1969). *Perceptrons : An Introduction to Computational Geometry*. The MIT Press, Cambridge (Mass.), London.

Mitchell, H. F. (1953). The Use of The Univ AC FAC-Tronic System in The Library Reference Field. *American Documentation*, 4(1):16–17.

Mitchell, T. (2006). The Discipline of mMchine Learning. Technical Report CMU ML-06 108, Carnegie Mellon ML Department.

Mitkov, R. (1999). Anaphora resolution: The state of the art. Technical report, School of Languages and European Studies, University of Wolverhampton.

Mladenić, D., Brank, J., Grobelnik, M., and Milic Frayling, N. (2004). Feature Selection Using Linear Classifier Weights: Interaction with Classification Models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 234–241, New York, NY, USA. ACM.

Mooers, C. (1961). From A Point of View of Mathematical etc. Techniques. In Fairthorne, R. A., editor, *Towards information retrieval*, pages xvii–xxiii. Butterworths, London, U.K.

Mooers, C. E. (1950). Coding, Information Retrieval, and the Rapid Selector. *American Documentation*, 1(4):225–229.

Mooers, C. N. (1959). The Next Twenty Years in Information Retrieval: Some Goals and Predictions. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), pages 81–86, New York, NY, USA. ACM.

Moranda, P. B. (1978). Surveyor's forum: Is software science hard? *ACM Computing Surveys (CSUR)*, 10(4):503–504.

Nanus, B. (1960). The Use of Electronic Computers for Information Retrieval. *Bulletin of the Medical Library Association*, 48(3):278–291.

National Institute of Standards and Technology (1991). *A Brief History of Measurement Systems, With A Chart of The Modernized Metric System*. U.S. Dept. of Commerce, National Institute of Standards and Technology.

Ng, V. (2008). Unsupervised models for coreference resolution. In *Procedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)*, pages 640–649, Honolulu, Hawaii, USA. ACL.

Ng, V. (2010). Supervised Noun Phrase Coreference Research: The First Fifteen Years. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1396–1411.

Ng, V. and Cardie, C. (2002). Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 104–111, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ng, V. and Cardie, C. (2003). Weakly supervised natural language learning without redundant views. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 94–101, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nguyen, N. L. T. and Kim, J.-D. (2008). Exploring domain differences for the design of pronoun resolution systems for biomedical text. In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 625–632, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nguyen, V., Deeds Rubin, S., Tan, T., and Boehm, B. (2007). A sloc counting standard. *COCOMO II Forum*, 2007.

Nikula, H. (1986). *Dependensgrammatik*. Ord och stil. LiberFörlag.

Nivre, J. (2005). Dependency Grammar and Dependency Parsing. Technical report, Växjö University.

Nivre, J. (2006). *Inductive Dependency Parsing (Text, Speech and Language Technology)*. Springer–Verlag New York, Inc., Secaucus, NJ, USA.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A Language–Independent System for Data–Driven Dependency Parsing. *Natural Language Engineering*, 13:95–135.

Nivre, J. and Nilsson, J. (2005). Pseudo–Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 99–106, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nouvel, D., Antoine, J.-Y., and Friburger, N. (2014). Pattern Mining for Named Entity Recognition. *LNCS/LNAI Series*, 8387i (post-proceedings LTC 2011).

Ogden, W. C. and Bernick, P. (1996). Oleada: User-centered tipster technology for language instruction. In *Proceedings of a Workshop on Held at Vienna, Virginia: May 6-8, 1996*, TIPSTER '96, pages 85–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

Orăsan, C., Cristea, D., Mitkov, R., and Branco, A. (2008). Anaphora resolution exercise: an overview. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Pak, A. and Paroubek, P. (2011). Text Representation Using Dependency Tree Subgraphs for Sentiment Analysis. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications*, DASFAA'11, pages 323–332, Berlin, Heidelberg. Springer–Verlag.

Park, R. E. (1992). Software size measurement: A framework for counting source statements. Technical Report CMU/SEI-92-TR-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Paroubek, P., Robba, I., Vilnat, A., and Ayache, C. (2008). EASY, Evaluation of Parsers of French: What Are The Results? In *Proceedings of the Sixth International Conference on Language Resources and Evaluation*, (LREC '08), Marrakech, Morocco. European Language Resources Association (ELRA).

Patwardhan, S. (2010). *Widening the Field of View of Information Extraction through Sentential Event Recognition*. PhD thesis, University of Utah.

Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2:559–572.

Peng, F., Ahmed, N., Li, X., and Lu, Y. (2007). Context Sensitive Stemming for Web Search. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 639–646, New York, NY, USA. ACM.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 433–440, Stroudsburg, PA, USA. Association for Computational Linguistics.

Petrov, S. and Klein, D. (2007). Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, pages 404–411.

Piskorski, J. and Yangarber, R. (2013). Information Extraction: Past, Present and Future. In Poibeau, T., Saggion, H., Piskorski, J., and Yangarber, R., editors, *Multi–source, Multilingual Information Extraction and Summarization*, Theory and Applications of Natural Language Processing, pages 23–49. Springer Berlin Heidelberg.

Poesio, M., Ponzetto, S. P., and Versley, Y. (2011). Computational models of anaphora resolution: A survey. *Linguistic Issues in Language Technology*.

Ponte, J. M. and Croft, W. B. (1998). A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA. ACM.

Ponzetto, S. P. and Strube, M. (2006). Exploiting semantic role labeling, wordnet and wikipedia for coreference resolution. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 192–199, New York City, USA. Association for Computational Linguistics.

Porter, M. F. (1980). An Algorithm for Suffix Stripping. *Program: Electronic Library and Information Systems*, 14(3):130–137.

Putnam, L. H. (1978a). Example of an early sizing, cost and schedule estimate for an application software system. In *The IEEE Computer Society's Second International Computer Software and Applications Conference*, COMPSAC '78, pages 827–832. IEEE Press.

Putnam, L. H. (1978b). A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, SE–4(4):345–361.

Radlinski, F. and Joachims, T. (2005). Query Chains: Learning to Rank from Implicit Feedback. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 239–248, New York, NY, USA. ACM.

Rahman, H. (2010). *Cases on Adoption, Diffusion, and Evaluation of Global E-governance Systems: Impact at the Grass Roots*. Premier Reference Source. Information Science Reference.

Robertson, S. (2004). Understanding Inverse Document Frequency: on Theoretical Arguments for IDF. *Journal of Documentation*, 60(5):503–520.

Robertson, S. and Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Robertson, S. E. (1977). The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4):294–304.

Robertson, S. E. (1997). Overview of The Okapi Projects. *Journal of Documentation*, 53(1):3–7.

Robertson, S. E. and Hancock Beaulieu, M. (1992). On the Evaluation of IR Systems. *Information Processing & Management*, 28(4):457–466.

Robertson, S. E. and Spärck Jones, K. (1976). Relevance Weighting of Search Terms. *Journal of The American Society for Information Science*, 27(3):129–146.

Robinson, J. J. (1970). Dependency Structures and Transformational Rules. *Language*, 46(2):259–285.

Rocchio, J. J. (1965). Relevance Feedback in Information Retrieval. Technical Report 9, Harvard University.

Rocchio, J. J. (1971). Relevance Feedback in Information Retrieval. In Salton, G., editor, *The Smart retrieval system – Experiments in Automatic Document Processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall.

Sabatier, P. and Pesant, D. L. (2013). Les Dictionnaires éectroniques de Jean Dubois et Françoise Dubois-Charlier et Leur Exploitation en TAL. In Gala, N. and Zock, M., editors, *Ressources Lexicales : Contenu, Construction, Utilisation, Évaluation*, volume Supplementa ; v. 30 of *Linguisticae investigationes*, pages 153–186. John Benjamins Publishing Company, Amsterdam, Netherlands.

Sager, N. (1960). A procedure for left to right analysis of sentence structure. *Transformations and Discourse Analysis Papers*, 27.

Sager, N. (1981). *Natural Language Information Processing: A Computer Grammmar of English and Its Applications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Sagot, B. and Fišer, D. (2008a). Building a free french wordnet from multilingual resources. In *Ontolex 2008*, Marrakech, Maroc.

Sagot, B. and Fišer, D. (2008b). Construction d'un Wordnet Libre du Français à Partir de Ressources Multilingues. In *TALN 2008 -Traitement Automatique des Langues Naturelles*.

Salton, G. (1968). *Automatic Information Organization and Retrieval*. McGraw Hill Text.

Salton, G. and Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management: An International Journal*, 24(5):513–523.

Salton, G., Wong, A., and Yang, C.-S. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620.

Salton, G. and Yang, C.-S. (1973). On The Specification of Term Values in Automatic Indexing. *Journal of Documentation*, 29(4):351–372.

Sample, T. and Hill, T. (1993). The architecture of a reverse engineering data model discovery process. *EDS Technical Journal*, 7(1).

Sanderson, M. and Croft, W. B. (2012). The History of Information Retrieval Research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451.

Schank, R. C. (1975). The Conceptual Approach to Language Processing. In *Conceptual Information Processing*, pages 5–21. North-Holland and Elsevier, Amsterdam and New York.

Schank, R. C. and Colby, K. M. (1973). *Computer Models of Thought and Language*. A Series of Books in Psychology. W. H. Freeman, San Francisco.

Schank, R. R. and Abelson, R. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–47.

Seddah, D., Candito, M., and Crabbé, B. (2009). Cross Parser Evaluation and Tagset Variation: A French Treebank Study. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 150–161, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sekine, S. and Nobata, C. (2004). Definition, dictionaries and tagger for extended named entity hierarchy. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC)*, Lisbon, Portugal. European Language Resources Association.

Serrano Guerrero, J., Olivas, J. A., Romero, F. P., and Herrera Viedma, E. (2015). Sentiment Analysis. *Information Sciences: An International Journal*, 311(C):18–38.

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Springer, Dordrecht, Netherlands.

Shaw, R. R. (1949). The Rapid Selector. *Journal of Documentation*, 5(3):164–171.

Shen, V. Y., Conte, S. D., and Dunsmore, H. E. (1983). Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Trans. Software Eng.*, 9(2):155–165.

Shen, V. Y. and Dunsmore, H. E. (1981). Analyzing cobol programs via software science. Technical report CSD TR-348, Department of Computer Sciences, Purdue University.

Shlens, J. (2014). A Tutorial on Principal Component Analysis. *CoRR*, abs/1404.1100.

Silva, G. and Dwiggins, D. (1980). Toward a prolog text grammar. *ACM Sigart Newsletter*, 73:20–25.

Smith, L. (1997). *Function Point Analysis and Its Uses*. Predicate Logic Inc.

Sneed, H. M. (2000). Extraction of Function-Points from Source-Code. In *Proceedings of the 10th International Workshop on New Approaches in Software Measurement*, IWSM '00, pages 135–146, London, UK. Springer-Verlag.

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.

Sørensen, T. (1948). *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons*. Biologiske Skrifter – Det Kongelige Danske Videnskabernes Selskab. I kommission hos E. Munksgaard.

Spärck Jones, K., editor (1981). *Information retrieval experiment*. Butterworths, London, Boston.

St Pierre, D., Maya, M., Abran, A., Desharnais, J.-M., and Bourque, P. (1997). Full Function Points: Counting Practices Manual. Technical Report 1997–04, Université du Québec à Montréal.

Steinhaus, H. (1956). Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801–804.

Stevens, M. E., editor (1965). *Statistical Association Methods for Mechanized Documentation: Symposium Proceedings*. U.S. Dep. of Commerce. National Bureau of Standards miscellaneous Publication. U.S. Government Printing Office.

Stroud, J. M. (1967). The fine structure of psychological time. *Annals of the New York Academy of Sciences*, 138(2 Interdiscipli):623–631.

Switzer, P. (1963). Vector Images in Document Retrieval. Technical Report 4, Harvard University.

Symons, C. R. (1988). Function point analysis: difficulties and improvements. *Software Engineering, IEEE Transactions on*, 14(1):2–11.

Tannier, X. (2012). WebAnnotator, an Annotation Tool for Web Pages. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey.

Tapanainen, P. and Järvinen, T. (1997). A Non-projective Dependency Parser. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, pages 64–71, Stroudsburg, PA, USA. Association for Computational Linguistics.

Taube, M., Gull, C. D., and Wachtel, I. S. (1952). Unit Terms in Coordinate Indexing. *American Documentation*, 3(4):213–218.

Taylor, R. W. and Frank, R. L. (1976). Codasyl data-base management systems. *ACM Comput. Surv.*, 8(1):67–103.

Tesnière, L. (1959). *Éléments de Syntaxe Structurale*. Éditions Klinksieck.

Tjong Kim Sang, E. F. (2002). Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING-02, pages 1–4, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, volume 4 of *CONLL '03*, pages 142–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Uemura, T., Kusumoto, S., and Inoue, K. (1999). Function point measurement tool for UML design specification. In *Proceedings of the Sixth International Software Metrics Symposium, 1999*, pages 62–69.

van Deemter, K. and Kibble, R. (2000). On coreferring: Coreference in muc and related annotation schemes. *Comput. Linguist.*, 26(4):629–637.

van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworth, 2nd edition.

van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition.

Vanni, M. and Zajac, R. (1996). The temple translator's workstation project. In *Proceedings of a Workshop on Held at Vienna, Virginia: May 6-8, 1996*, TIPSTER '96, pages 101–106, Stroudsburg, PA, USA. Association for Computational Linguistics.

Verhoeff, J., Goffman, W., and Belzer, J. (1961). Inefficiency of the Use of Boolean Functions for Information Retrieval Systems. *Communications of the ACM (CACM)*, 4(12):557–558.

Voorhees, E. M. (2001). The TREC Question Answering Track. *Natural Language Engineering*, 7(4):361–378.

Walter, C. (2005). Kryder's law. *Scientific American*, 293(2):32–33.

Wang, T., Viswanath, V., and Chen, P. (2015). Extended Topic Model for Word Dependency. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 506–510, Beijing, China. Association for Computational Linguistics.

Whitmire, S. A. (1995). An Introduction to 3D Function Points. *Software Development*, 3(4):43–53.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83.

Wilks, Y. (1987). Text searching with templates. Technical Report Technical Report ML 162, Language Research Unit, Cambridge University.

Wilks, Y. (1997). Information extraction as a core language technology. In *International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, SCIE '97, pages 1–9, London, UK. Springer-Verlag.

Wilks, Y. A. (1973). *Preference Semantics*. Memo (Stanford Artificial Intelligence Laboratory). Computer Science Department, Stanford University.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press, Inc., Orlando, FL, USA.

Yang, X., Zhou, G., Su, J., and Tan, C. L. (2003). Coreference resolution using competition learning approach. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 176–183, Sapporo, Japan. Association for Computational Linguistics.

Yang, Y. and Liu, X. (1999). A Re–examination of Text Categorization Methods. In *Proceedings of the* 22^{nd} *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 42–49, New York, NY, USA. ACM.

Yeh, A. (2000). More Accurate Tests for the Statistical Significance of Result Differences. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 2*, COLING '00, pages 947–953, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zarri, G. P. (1983). Automatic representation of the semantic relationships corresponding to a french surface expression. In *Proceedings of the First Conference on Applied Natural Language Processing*, ANLC '83, pages 143–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least–Effort*. Addison–Wesley, Cambridge, MA.

Zweben, S. H. (1979). Surveyor's Forum: Heads I Win, Tails You Lose. *ACM Computing Surveys (CSUR)*, 11(3):277–278.

Zwicky, A. M. (1985). Heads. *Journal of Linguistics*, 21:1–29.

Author's Publications

Munshi Asadullah, Damien Nouvel, and Patrick Paroubek. "Using Verb-Noun Pat-terns to Detect Process Inputs". In Proceedings of the Text, Speech and Dialogue – 17th International Conference, TSD 2014, Brno, Czech Republic, September 8–12,2014. Pp. 181–188. 2014

Munshi Asadullah, Patrick Paroubek, and Anne Vilnat. "Bidirectionnal converterbetween syntactic annotations : from French Treebank Dependencies to PASSAGE annotations, and back". In Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, May 26–31, 2014. Pp. 2342–2347.

Munshi Asadullah, Patrick Paroubek, and Anne Vilnat. "Converting from the French Treebank Dependencies into PASSAGE syntactic annotations". In Proceedings of the 6th Language Technology Conference : Human Language Technologies as a Challenge for Computer Science and Linguistics (LTC 2013), Poznan, Poland, December 7–9, 2013. Pp. 188–182. 2013

Patrick Paroubek, **Munshi Asadullah**, and Anne Vilnat. "Convertir des analyses syntaxiques en dépendances vers les relations fonctionnelles PASSAGE". In Actes dela 20ème Conférence sur le Traitement Automatique des Langues Naturelles (TALN2013), Les Sables d'Olonne, France, June 17–21, 2013. Pp. 675–682.

Appendix A

Miscellaneous

A.1 Annotation Guideline

There are *four* types of elements defined in the guideline for annotation as summarized in Table A.1,

Table A.1. Annotation Elements

Category	Description
FUN_GRP	Functional Groups: grouping of functional elements for easier counting.
APP	External Applications
FUN	Transaction Functions
DATA_GRP	Data Functions

The guidelines provide a general framework for annotating any specification document for FP identification. The important aspects of these guidelines are listed below,

- The use of either *MicroSoft* Word or *OpenOffice* Writer.
- Highlight only the part of text "necessary and sufficient" for the identification of FPs.
- The text part containing the information must be highlighted seamlessly i.e. uninterrupted character span should be present.
- Each highlighted part must at least contain a noun (common or proper) or a verb.
- The text must not be changed (content or format).
- Preserve the original document format for archiving.

We have made one significant change in the guidelines this year. We have adopted the use of HTML format and the open–source annotation tool *WebAnnotator* (Tannier, 2012) instead of office tools (*Microsoft* or *OpenOffice*). *WebAnnotator* is a tool for annotating Web pages, implemented as a *Firefox* extension, allowing annotation of both offline and online *HTML* contents. A predefined *DTD* can be used to define the visualization of the data that can use different highlighting colours for different types of annotation. Furthermore, the output *HTML* content contains all the formatting we are interested to preserve for text extraction. The annotations do not interfere with the original text since they overlay the annotation information on top of the original text. The tool has been proven to be easy to learn to use with minimum training and was integrated in our processing chain easily.

A.2 ProjEstimate Corpus DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="DOCUMENT" type="DOCUMENTType" />
  <xsd:complexType name="DOCUMENTType">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" name="S" type="SType" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SType">
    <xsd:sequence>
      <xsd:element name="T" type="TType" />
      <xsd:element name="DEP" type="DEPType" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" />
    <xsd:attribute name="page" type="xsd:int" />
  </xsd:complexType>
  <xsd:complexType name="DEPType">
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="src" type="xsd:string" />
```

```
    <xsd:attribute name="target" type="xsd:string" />
  </xsd:complexType>
  <xsd:complexType name="TType">
    <xsd:sequence>
      <xsd:element name="SURF" type="xsd:string" />
      <xsd:element name="LEMA" type="xsd:string" />
      <xsd:element name="GPOS" type="xsd:string" />
      <xsd:element name="SPOS" type="xsd:string" />
      <xsd:element name="MORP" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" />
    <xsd:attribute name="annot" type="xsd:string" />
  </xsd:complexType>
</xsd:schema>
```

A.3 Lloyd's Algorithm

Consider a set $X = \{x_1, x_2 \ldots x_n\}$, and distance $d : X \times X \rightarrow \mathbb{R}+$ and the output is a set $C = \{c_1, c_2 \ldots c_k\}$ i.e. the set of centroids. This implicitly defines a set of clusters where $\phi\ C(x) = \text{argmin}_{c \in C}\ d(x, c)$

Lloyd's Algorithm for k-Means Clustering

Choose k points $C \subset X$ (arbitrarily*)
repeat
    For all $x \in X$ find $\phi\ C(x)$ (closest center $c \in C$ to x)
    For all $i \in [j]$ let $c_i = \text{average}\{x \in X \mid (x) = c_i\}$
until The set C is unchanged

A.4 Complete Feature Types for ML

Every unique pattern for each feature type has been used as a feature. The name contains the clue about the pattern. The prefix contains the lexical entity used as the pattern and the suffix either expresses the n-gram type or d-gram type used for the pattern. There are 5 lexical groups that has been used,

1. **Surface Form** (`surface`): The actual extracted token (with inflictions and other features) from the data files.
2. **Lemma** (`lemma`): The lemma for each token.
3. **General POS** (`gpos`): The general POS group of each token as defined in the FTB Annotation Guideline[1].
4. **Specific POS** (`spos`): The detailed POS description that includes additional morphological information also defined in the FTB Annotation Guideline.
5. **Lemma and General POS** (`lemma_gpos`): The lemma and the POS put together with a front slash character ("/"), e.g. `book/n` is the representation of the *noun* lemma "*book*".

Along with these lexical features 4 different *n-gram* representations for all the groups have been used and represented as the suffix `unigram`, `bigram`, `trigram` and `quadgram` for the value of `n=1, 2, 3 and 4` respectively. There are also 3 d-gram patterns generated designated as the suffixes `dgram`, `dgram_wild` and `extended_dgram` using the concepts of *d–gram*, *d–gram with wild card* and *extended d–gram* (see § **??**). Many of these features are expected to show little or no patterns when applied to the data, however, the objective is to cover all our basics before choosing the right set of features.

A.4.1 Feature List

```
surface_unigram
surface_bigram
surface_trigram
surface_quardgram
surface_dgram
surface_dgram_wild
surface_extended_dgram
```

[1] http://alpage.inria.fr/statgram/frdep/Publications/FTB-DescriptionDepSurface.pdf

```
lemma_unigram
lemma_bigram
lemma_trigram
lemma_quardgram
lemma_dgram
lemma_dgram_wild
lemma_extended_dgram

gpos_unigram
gpos_bigram
gpos_trigram
gpos_quardgram
gpos_dgram
gpos_dgram_wild
gpos_extended_dgram

spos_unigram
spos_bigram
spos_trigram
spos_quardgram
spos_dgram
spos_dgram_wild
spos_extended_dgram

lemma_gpos_unigram
lemma_gpos_bigram
lemma_gpos_trigram
lemma_gpos_quardgram
lemma_gpos_dgram
lemma_gpos_dgram_wild
lemma_gpos_extended_dgram
```

Appendix B

Measurement Scales

B.1 Nominal Scale

Nominal scale is a rather primitive form of measurement, where each entity is placed in a class or category based on the value of the attribute of our interest. The empirical relation system in this type of scales consist only of discrete classes but there is no notion of ordering among the classes. Any distinct numbering or symbol assignment for the classes is enough, without the notion of magnitude thus no implied notion of comparison, rather a contrast between entities in terms of the desired attribute can be established. For example, a company produces cars of three colours (e.g. red, black and blue) thus, the following mappings are equally capable of representing relation,

$$M_1(x) = \begin{cases} 7\text{: if x is a “red” car} \\ 2\text{: if x is a “black” car} \\ 9\text{: if x is a “blue” car} \end{cases} \qquad M_2(x) = \begin{cases} ?\text{: if x is a “red” car} \\ !\text{: if x is a “black” car} \\ +\text{: if x is a “blue” car} \end{cases}$$

The scale in the example is establishing the colour for each car being produced and no other quantification or qualification.. It can easily be seen that it is a simple one–to–one mapping and any two mapping M and M' will always be related, i.e. M' can be obtained from M by a simple remapping. Since the simplest mathematical manipulation such a ordering can not be performed, any number or any symbol will suffice. It is also clear that the class of admissible transformation for a nominal scale measure is the set of all possible one–to–one mapping.

B.2 Ordinal Scale

Ordinal scale is an augmentation on the nominal scale with the information about some sort of ordering of the classes or categories in the empirical relation system thus allow us to perform analysis that is not possible using the nominal scale. The classes are ordered with respect to the attribute of interest, any mapping that preserves the ordering is accepted and the numbers only represent the ordering thus any arithmetic operation (i.e. addition, subtraction etc.) have no meaning. For example, if we are looking to establish a scale for complexity of math problems with four possible levels, *trivial*, *simple*, *moderate*, *complex*, *impossible*. The complexity is expected to be higher for a level from the previous one and thus the mapping cannot be done as freely as with nominal scale. Numbers are more appropriate for this mapping and the only thing we have to make sure is that higher level of complexity need to be assigned a higher number. All the following mappings are valid,

$$M_1(x) = \begin{cases} 1: \text{if x is trivial} \\ 2: \text{if x is simple} \\ 3: \text{if x is moderate} \\ 4: \text{if x is complex} \\ 5: \text{if x is impossible} \end{cases} \quad M_2(x) = \begin{cases} 2: \text{if x is trivial} \\ 5: \text{if x is simple} \\ 6: \text{if x is moderate} \\ 7: \text{if x is complex} \\ 9: \text{if x is impossible} \end{cases} \quad M_3(x) = \begin{cases} 0.001: \text{if x is trivial} \\ 0.01: \text{if x is simple} \\ 0.1: \text{if x is moderate} \\ 59.2: \text{if x is complex} \\ 92.73: \text{if x is impossible} \end{cases}$$

Since the principal is to preserve the ordering of the classes, formally it is to map a set of ordered classes $\{C_1, C_2, C_3, \ldots C_n\}$ into a increasing series of numbers $\{x_1, x_2, x_3, \ldots x_n\}$ where $x_i > x_j$ when i is grater than j. The admissible transformation is the set of what is called a *monotonic mapping* and any increasing series of length n can be substituted for the value set.

B.3 Interval Scale

Ordinal scale conveys more information about an entity than the nominal scale, interval scale carries further information, making it a more powerful representation for the empirical relations. In this scale the values not only represent the order but also the size of the interval reflects for far each value is from one another.The difference only preserves the distance between values but not the ratio thus calculating ration for values in such a scale does not make any sense. Thus, some arithmetic operations namely, addition and subtraction is accepted but multiplication and division are not acceptable. The temperature scales can be a

proper example,

> We can measure temperature in several scales such as, Fahrenheit or Celsius scale. The values maintain order i.e. 20°C is higher than 19°C and also represent the change in temperature between two values having the same difference at any range i.e. the increase of heat for the increase of temperature from 20°C to 25°C is the same when the temperature increased from 50°C to 55°C. However we cannot make inference about the ratio of temperature, i.e. an air temperature of 50°C is not twice as hot as 25°C.

Formally, if an attribute is measurable on an interval scale, e.g. temperature, and M and M' being two possible mappings that satisfy the representation conditions e.g. Celsius and Fahrenheit, there exist two numbers a and b such that, $M = aM' + b$. This type of liner transformation also known as *affine transformation* and for the temperature scales for Celsius and Fahrenheit the transformation function is $F = 9\Big/5C + 32$.

B.4 Ratio Scale

If we are looking for more information on the empirical relations, for example if we want to know if a liquid is twice as hot as another liquid, we have to shift out mapping to a ratio scale. This is a very important scale in physical science and most common scale in use. This measurement preserves ordering, size of interval between entities and the ration between entities. There must be a zero element representing the total lack of an attribute, then increase in equal intervals (i.e. *units*). All arithmetic operations are meaningful while applied to the classes in the ranges of the mapping. Formally, a transformation for a ratio scale between two mapping is in the form of $M = aM'$ where a is a positive integer and these type of transformation is called ration transformation because it preserves the ratio.

From our previous example of interval scale, Celsius and Fahrenheit scales are not ratio scales because, 0°C or 0°F does not imply the absence of heat whereas the *Kelvin(°K)* scale for temperature is a ratio scale because 0°K actually implies total lack of heat. Moreover, the lack of meaningful ratio in the scales like Celsius, i.e. temperature of 20°C does not imply twice as much heat present than 10°C, although, converting them to Kelvin scale shows us, the ratio between 283°K (10°C) and 293°K (20°C) is actually *1.033*. Another example is the length measurement of physical object in any unit, all possible units preserve the ratio between the length of two entities. Zero length however, only exists in the limit of things getting smaller and smaller and more at the conceptual realm rather than hard physical reality.

B.5 Absolute Scale

The more information the scale of measurement carry, the more restrictive the set of admissible transformation become. Absolute scale is the most restrictive form of scale, for any two measurements M and M' there is only one admissible transformation, the identity transformation. The measurement in this scale is made by counting the number of elements in the entity set, so, the attribute always take the form of number of occurrence of "x". There is only one measurement mapping, i.e. the count and all arithmetic analysis of the resulting count is meaningful.

A simple example could be counting number of tokens, a very common task in text processing. Let us presume there are n tokens in a text file ($file_n$), thus, it allows us to infer statements such as the size of the file in terms of tokens is n. It also allow us to compare other files, e.g. if there is another file with m tokens ($file_n$) the following statements are all meaningful,

- $file_m$ is bigger than $file_n$ if $m > n$
- total token is $m+n$ or the difference is $m-n$
- the ratio of tokens is m/n
- if $n = 2m$ $file_m$ is half the size of $file_n$

However, the same statements cannot be made about, someone's age, i.e. measurement of age is not absolute because it can be measured in year, month, day, hour or seconds thus the identity transformation argument is not valid, thus it is in ratio scale.

B.6 Statistical Meaningfulness

Association of statistical meaningfulness with respect to different types of scale is an important aspect of measurement. Many statistical analyses use basic arithmetic operations (e.g. $+, -, \times, \div$ etc.) and we are implying basic statistics such as measure of central tendency i.e. *mean*, *median*, *mode* etc. or measure of dispersion i.e. *variance* or *standard deviation* that gives an indication of the whole dataset and the distribution. These simple analytical techniques however, are meaningful only within the context of certain scale types. For example, the computation of mean, variance and standard deviation convey no meaning for the nominal and ordinal types. On the other hand, median is meaningful for ordinal type measure, e.g. let us consider the mappings in Table B.1 for our old math complexity problem,

Table B.1. Math Problem Complexity Scale Maps

	trivial	**simple**	**moderate**	**complex**	**impossible**
M	1	2	3	4	5
M'	1	2	3	4	9
M''	0.77	3.9	56	114	478

Let us consider 2 sets of math problems, X and Y, defined as,

$$X = \{\textit{trivial, simple, moderate, simple, impossible}\}$$
$$Y = \{\textit{moderate, simple, complex, complex, complex, moderate, moderate}\}$$

The mean and median for the sets using the 3 mappings have been summarized in Table B.2. It can be clearly seen that with the mapping M the mean for X is less than the mean for Y but with the mapping M' the relation is no longer true. However, the median values maintain the relation i.e. $X_{median} < Y_{median}$, throughout, even when we mapped it with a drastically different mapping M'', the relation holds. Thus, establish the meaningfulness of median for

Table B.2. Mean and Median Summary for The Datasets

	Mean		***Median***	
	X	**Y**	**X**	**Y**
M	***2.6***	***3.28***	2	3
M'	***3.4***	***3.28***	2	3
M''	123.76	73.41	3.9	56

measuring central tendency in ordinal scale data and not that of mean. Mean can be used as a meaningful measure for interval and ration scale data. Formally, if we need to prove that two datasets $x_1, \ldots, x_n$ and $y_1, \ldots, x_m$ measured in a ratio scale having the means x_{mean} and y_{mean} maintain the relation $x_{mean} > y_{mean}$ is meaningful, we must demonstrate that,

$$\frac{1}{n}\sum_{i=1}^{n} M(x_i) > \frac{1}{m}\sum_{j=1}^{m} M(x_j) \quad \textit{if and only if} \quad \frac{1}{n}\sum_{i=1}^{n} M'(x_i) > \frac{1}{m}\sum_{j=1}^{m} M'(x_j)$$

The ratio scale gives us the needed information in the form of *admissible transformation* i.e. $M = aM'$ for some $a > 0$. When we substitute aM' for M, in the above equation, we get a

Table B.3. Summary of Statistical Analysis and Their Meaningfulness to Scale Type

Scale Type	Defining Relations	Appropriate Statistics	Appropriate Test
Nominal	Equivalence	Mode Frequency	Non–Parametric
Ordinal	Equivalence Greater Than	Median Percentile Spearman r Kendall r Kendall W	Non–Parametric
Interval	Equivalence Greater Than Ratio of Two Intervals	Mean Standard Deviation	Non–Parametric
Ratio	Equivalence Greater Than Ratio of Two Intervals Ratio of Two Values	Geometric Mean Coefficient of Variation	Non–Parametric Parametric

statement that is clearly valid. The same principle applies for any statistical technique, using scale and transformation properties to verify that a certain analysis is valid for a given scale type. Table B.3 presents a summary of meaningful statistics for different scale types, the statistical analyses is inclusive while reading downward. So, the statistical analysis valid for ordinal scale is also valid for all the other scale types with higher information content.

Appendix C

The Theory of Software Science

C.1 Proposed Metric

1. n_1 : number of unique operators
2. n_2 : number of unique operands
3. N_1 : number of total operators
4. N_2 : number of total operands

C.1.1 Program Length (N) and Vocabulary Size (n)

The length of the program is defined as the total number of operator and operands, whereas, the size of the vocabulary is defined as the total number of unique operator and operands. Formally,

$$N = N_1 + N_2 \qquad \textit{and} \qquad n = n_1 + n_2$$

C.1.2 Volume (V), Program Level (L) and Difficulty (D)

Volume is the actual size (measured in *bits*) of a program in a computer, given that a uniform binary encoding has been used and defined as,

$$V = N \times \log_2 n$$

Since, an algorithm is expected to be implemented in different but equivalent programs, the program with the minimal size is said to have the *Potential Volume* (V^*). Any given program with volume V is considered to be implemented at the *Program Level L* defined as,

$$L = V^* \Big/ V$$

The value of L ranges between zero and one and $L = 1$ can be interpreted as a program written in the highest possible level (i.e. with minimum size). The difficulty of a program is in an inverse relation with the difficulty. Formally,

$$D = 1 \Big/ L$$

So, as the *volume* of an implementation of a program increases, the *program level* decreases and thus, the *difficulty* increases. For example, if the same operands are used many times in the program, it will increase volume and difficulty, thus make it more prone to errors.

C.1.3 Effort to Implement (E)

The effort is defined to be proportional to the size of the program, i.e. larger program requires more effort. It also takes more effort if the program is at a lower level (i.e. higher difficulty) in comparison to an equivalent program at a higher level (i.e. lower difficulty). The unit of effort measure was defined as *number of elementary mental discriminations*,

$$E = V \Big/ L = V \times D$$

C.1.4 Length Equation

The first hypothesis of *Software Science* is that the length of a program is a function of the number of unique operator and unique operand only. In practice, the metric may not be very precise for a given program but can be considered valid from a statistical point of view.

$$\hat{N} = n_1 \times \log_2 n_1 + n_2 \times \log_2 n_2$$

C.1.5 Potential Volume

We discussed earlier that if a program is implemented at its highest possible level i.e. lowest possible size it will have the *potential volume* V^*. A practical scenario is, if a desired operator is already defined in a library, i.e. dealing with a function call from a library, V^* is achieved by the name of the procedure and the list of input/output parameters. The vocabulary consists of two *operators* and n_2^* operands. One operator is the name of the function and the other is the grouping symbols (e.g. parenthesis). Thus, it will be defined as,

$$V^* = \left(2 + n_2^*\right) \times \log_2 \left(2 + n_2^*\right)$$

This hypothesis however, is not applicable universally since there are programs that do not have explicit input/output parameters (e.g. a compiler's output consist of arbitrary number of files and massages to the operating system). Halstead (1977), himself made the observation that, the concept of n_2^* was inadequate for high information content *constants* and other implicit variables.

C.1.6 Program Level and Difficulty Estimator

The program level of an implementation depends on the ratio of the *potential volume* and the actual *volume*, but, *potential volume* is not always available and so an alternate for estimating the level had be proposed,

$$\hat{L} = \frac{1}{\hat{D}} = \frac{2}{n_1} \times \frac{n_2}{N_2}$$

As it can been seen, in this equation the difficulty increases if either additional operators are introduces (i.e. the $\frac{n_1}{2}$ term increases) or an operator has been used more times (i.e. the $\frac{N_2}{n_2}$ term increases) or both. It must also be noticed that all the parameter are obtained from counting just the operators and the operands.

C.1.7 Programming Time

Another important claim of *Software Science* was to relate the basic metrics with actual implementation time. John Stroud (Stroud, 1967) claimed that mind is capable of making a limited number of elementary discriminations per second. He claimed that this number S (known as the *Stroud Number*) ranges between 5 and 20. Thus, when the earlier mentioned

Effort (E) being divided by S will give us the programming time in seconds,

$$T = E \Big/ S$$

S is normally set to 18 since that had given Halstead (1977) the best results during his experiments. He claimed that the equation can be used to estimate programming time when a given problem is solved by a single, proficient, concentrating programmer writing a single-module program.

C.1.8 The Language Level

The language level is the metric to express the strength or weakness of a programming language. Halstead (1977) hypothesized that if the programming language is kept fixed than as V^* increases L decreases in such a way the product $L \times V^*$ remains constant. This constant (called the language level λ) thus, can be used to characterize a programming language,

$$\lambda = L \times V^* = L^2 V$$

Shen et al. (1983) presented that, analyses were made on a number of programs written on different languages and the following determination was made on their language level,

- 1.53 for PL/l
- 1.21 for AlgoL
- 1.14 for Fortran, and
- 0.88 for CDC assembly language

They also argued that these values also followed most programmers' intuitive rankings for these languages, but they have large variances. The observation seems quite reasonable since, the size of a program or the opinion about a language depends on many factors including the problem being solved, the programmers skill and nevertheless personal preferences. However, while comparing different programming languages, λ can be useful if a set of problems are programmed by the same programmer in different languages. The formula can also be used to derive the formula for effort E,

$$E = V^{*3} \Big/ \lambda$$

According to this formula for a given problem, the Effort and consequently, the resulting implementation time varies according to the squared inverse of the Language Level.

C.2 Criticism

Regardless of having the appearance of a sound theory, *Software Science* has been heavily criticized in contemporary publications (Fenichel, 1979; Lassez et al., 1981; Malenge, 1980; Moranda, 1978). Some of the criticisms were about the soundness of the theory and others about the empirical results that failed to support the theory. Shen et al. (1983) presented that, one of the primary concern was regarding the definition and counting guideline for the operators and operands. The original theory was developed for analyzing algorithms rather than programs. Most supporting data was also drawn from algorithms written in *Algol* and *Fortran*. It was quite liner to define an algorithm to be a collection of operator and operands and classify different tokens for *Algol* or *Fortran* into operator and operands. But, it was found to be more difficult to do so for programs especially, in other languages. Lassez et al. (1981) argued that it is sometimes impossible to determine whether a token is an operator or an operand since the classification may only be known at runtime or may depend on the declaration parameters. Malenge (1980) argued that even Halstead (1977) produced some inconsistency while counting these elements. The original method ignores all variable declarations, however, several authors (Elshoff, 1978; Fitsos, 1979; Shen and Dunsmore, 1981) argued that in many languages (e.g. Cobol) variable declaration is a significant part of the programming effort thus, ignoring them do not seem reasonable.

The other significant complain about the theory was regarding the actual derivation of the equations. Shen et al. (1983) argued that in many of the derivations, several implied assumptions were made without providing any theoretical justification. For example, while deriving the length equation, the author divides a program of length N into $N \Big/ n$ sub–strings of length n and considering, there are no duplicate of these sub–strings and operators and operands alternate, Halstead (1977) concluded that the following inequality must be satisfied,

$$N < n_1^{n_1} \times n_2^{n_2}$$

However, no explanations were provided for dividing a program into $N \Big/ n$ sub–strings. Moreover, the assumption that operators and operands alternate implies that for all program $N_1 \approx N_2$, but Shen et al. (1983) argued that the general observation suggested otherwise.

Furthermore, the above inequality equation assumes that an operator or an operand will appear first, but doing so will double the upper limit, thus, the Length equation cannot be justified on theoretical grounds. Another derivation problem appears while Halstead (1977) derived the *boundary volume* V^{**} as,

$$V^{**} = \left(2 + n_2^* \times \log_2^{n_2^*}\right) \times \log_2\left(2 + n_2^*\right)$$

and later he sets,

$$\frac{dn}{dn_1} = \frac{V^{**}}{V^*}$$

and no justification for the formula was provided and treating the discrete variable n as continuous and take derivation is at least questionable.

Then again, while deriving the time equation, he relied heavily on the work of Stroud (1967), but among psychologists there is no general acceptance of the hypothesis thus, as a theoretical concept, the *time equation* must therefore be considered pure assumption. The presence of these unverifiable assumptions make the underlying theoretical foundations of *Software Science* rather shaky if not seriously flawed.

One may argue that even though the theoretical foundation of *Software Science* is weak, yet it may be useful, at least, for statistical analysis, but the empirical work has been criticized as well. Zweben (1979) concluded that the validating data reported in (Halstead, 1977) and in some early papers that followed were not presented in the classical form of hypothesis testing. Shen et al. (1983) resonating with the observation presented that Halstead frequently and incorrectly inferred that because two sets of numbers were highly correlated, they can be substituted for each other.

Moreover, the general practice of the empirical experiments were criticized. First of all, the sample size used to make inferences were small (less than 10 in contrast to the general practice of having a sample size of between 20 and 30). Then, the programs involved were small (all except one were single module and less than 50 statements) so, prediction over scaled projects with multi-module was nevertheless far fetched. There were also the criticism about the subjects used for the experiment (most experiments had only a single subject and even when there were more, most of them are college students), but we would like to express our appreciation considering a pioneering work done in an university environment. Later works provided better analysis with better dataset and regardless of the already discussed flaws the theory found to be quite useful. For example, there is overwhelming evidence using existing analyzers to suggest the validity of the *Length Equation* in several languages (Shen

et al., 1983).

$$\hat{N} = n_1 \times \log_2^{n_1} + n_2 \times \log_2^{n_2} \approx n \log_2 \frac{n}{2}$$

Thus, the *Length Equation* is a function of the count of the unique basic tokens and miscounting of an operator or an operand has virtually no effect. It may effect significantly though, if there are many unique tokens that have been used for a small number of times.

Appendix D

Value Adjustment Factor

***V**alue **A**djustment **F**actor (VAF)* adjusts the final FP count on the basis of 14 features known as the ***G**eneral **S**ystem **C**haracteristics (GSC)* and may vary the final count upto $\pm 35\%$. The 14 GSC definitions[1] can be as follows,

1. ***Data communications*** represents the degree to which an application communicates for data. It could be local i.e. batch process based stand alone application mostly communicate with local resources, or on–line i.e. heavily relay on one or more protocol based remote communication.
2. ***Distributed data processing*** describes the degree to which the application transfers data among physical components of the application. It is a characteristic of the application within the application boundary.
3. ***Performance*** represents the level of response time or throughput considerations influenced the application development.
4. ***Heavily used configuration*** is the degree to which computer resource restrictions influenced the development of the application e.g. the user may want to run the application on existing or committed equipment that will be heavily used.
5. ***Transaction rate*** describes the degree to which the rate of business transactions influenced the development of the application. The transaction rate is high, and it influences the design, development, installation, and support of the application.
6. ***On–Line data entry*** represents the portion of data being entered or retrieved through interactive transactions e.g. On-line User Interface for data entry, control functions, reports, and queries that are provided in the application.
7. ***End–user efficiency*** describes the degree of consideration for human factors and ease of use for the user of the application.

[1] http://www.functionpointmodeler.com/fpm-infocenter/index.jsp

8. ***On–Line update*** describes the degree to which internal logical files are updated on–line.
9. ***Complex processing*** describes the degree to which processing logic influenced the development of the application.
10. ***Re–usability*** describes the degree to which the application and the code in the application have been specifically designed, developed, and supported to be usable in other applications.
11. ***Installation ease*** describes the degree to which conversion from previous environments influenced the development of the application.
12. ***Operational ease*** describes the degree to which the application attends to operational aspects, such as start–up, back–up, and recovery processes.
13. ***Multiple sites*** describes the degree to which the application has been developed for different hardware and software environments.
14. ***Facilitate change*** describes the degree to which the application has been developed for easy modification of processing logic or data structure.

Each of these GSC is then associated with a ***D**egree of **I**nfluence (DI)* having a value between 0 and 5 and it has been often described as presented in Table D.1,

Table D.1. DI Level Definition

Level	Definition
0	Not present, or no influence
1	Incidental influence
2	Moderate influence
3	Average influence
4	Significant influence
5	Strong influence throughout

Once these assessments are made the DI of the application and the ***T**echnical **C**omplexity **F**actor (TCF)* is calculated using the following formulas,

$$DI = \sum_{i=1}^{14} GSC[i]$$

$$TCF = 0.65 + 0.01 \times DI$$

Thus each DI contributes about 1% to the value of TCF and can range between 0.65 and 1.35. Finally, the adjusted or reported FPs can be calculated using the following formula.

$$FP = UFP \times TCF$$

Appendix E

Variations in Methods and Counting Practice

The original method and the considerations for FP counting (Albrecht, 1979) have been modified significantly over the last 45 yeas, but the the original idea is still at the core of most if not all FPA method. Several updates were proposed during this time and some eventually became ***I**nternational **S**tandardization **O**rganization (ISO)* certified methods. Capers Jones (1991) listed 20 variants of the functional metrics including the ISO recognized variants. In all practical purposes, the big four variants i.e. IFPUG, COSMIC, MARK II and NESMA approaches account for 97% of the total usage (Jones, 1991). They all have formal training and certification programs thus, produce a more homogeneous counting practice. In the following section we shall highlight the similarities and differences in some of the significant methods with respect to the general practice suggested by the IFPUG–CPM. Many of the methods mentioned by Jones (1991) either never gain main stream popularity or proven more hazardous than useful over the development period.

E.0.1 MARK II (MkII) Function Point

Symons (1988) proposed the ***MkII Function Point Method***, aimed to improve on Albrecht's approach by taking into account the internal complexity of data–rich business application software, which is also one of the 5 ISO recognized methods. The method is heavily based on Albrecht's original method, however the software environment model used in MkII is different. MkII FPA retains the fundamental premise that the function point count obtained

by the method is the product of the two components, ***I**nformation **P**rocessing **S**ize (IPS)* and the ***T**echnical **C**omplexity **A**djustment (TCA)*. The TCA component (i.e. TCF and DI) is mostly unchanged from that of Albrecht's but the most notable difference is the extension of the list of GSC from fourteen to nineteen or more factors. The functional model used for the actual counting was also rather different since instead of the 5 distinct types functional unit, in MkII a system is comprised of only discrete logical transaction. These transactions are as Symons (1988) presented, are "*a unique input/process/output combination triggered by a unique event of interest to the user, or a need to retrieve information*".

Using this definition, identifying unique transactions is quite simple when a system is developed using methods such as, ***S**tructured **S**ystems **A**nalysis and **D**esign **M**ethod (SSADM)*[1] (Great Britain Treasury and Telecommunications Agency, 2000). Symons (1988) even claims that once the idea of the logical transaction is completely understood, even sizing a installed systems is "perfectly straightforward". Formally put, if t is the unadjusted function point equivalent and a transaction is comprised of a set of input data element type (I), a set of entity type referenced (E) and a set of output data element types (O) and a set of three weighting factors in the same order for the corresponding elements $\{W_i, W_e, W_o\}$ and the system is comprised of n such transactions the UFP can be determined using the following formula,

$$UFP = \sum_{i=1}^{n} t[i] \qquad \text{given,}$$
$$t = W_i \times count(I) + W_e \times count(E) + W_o \times count(O)$$

The weights W_i, W_e and W_o are not changeable but can be subject to calibration and the changes to the weights are published, in the first instance, through the ***U**nited **K**ingdom **F**unction **P**oint **U**ser **G**roup **L**imited (UKFPUG)*[2]. MkII guideline thus, suggests calculating the TCA next and the difference with Albrecht's approach is that instead of using a fixed 1% influence of each DI, a coefficient C is used and DI is calculated from nineteen GSC. C is calculated during calibration and the TCA is calculated using the following formula.

$$TCA = 0.65 + C \times DI$$

The MkII method allows for revision or calibration of the relative weightings of the individual components of the TCA i.e. C and transaction component weights. According to MkII FPA

[1] The latest version of SSADM includes the MkII FPA Estimating Method to assist in the production of reliable project estimates.

[2] http://www.uksma.co.uk/

the purpose of sizing is to help measure productivity, therefore, the relative weights should reflect the relative average effort needed to analyse, design and develop the software. The calibration process is a data driven and somewhat subjective process that starts with gathering data on two factors, information processing component (X) and technical complexity adjustment (Y). People are asked to give a ratio of effort assigned by an individual familiar with the software development process in terms of work hours (e.g. 70% to 30% or 80% to 20% etc.). The actual TCA value is then deduced using the following formula,

$$TCA_{actual} = 0.65 \times \left(1 + \frac{Y}{X}\right)$$

The value 0.65 was used to make it comparable to the derived TCA form the DI. Once a acceptable number of projects have been examined, the data can be plotted in a graph with actual TCA on the horizontal axis and the computed TCA on the vertical axis and then C is computed by the forced best fit between these two sets of data. It was found that the best fit is 0.005, half that had been used that of Albrecht's value of 0.01. The intuition behind the idea is that, since the time of the original FPA, some of the GSC became less relevant (e.g. *data backup* which is often an automatic process and a de facto feature in modern systems, or cross–system communication improved radically in the modern systems thanks to improved hardware and protocols) i.e. a developer does not have to explicitly take care of them, One can even envisage that at some point in the future the all GSC may need to be revised since new complexities are regularly added in the development process.

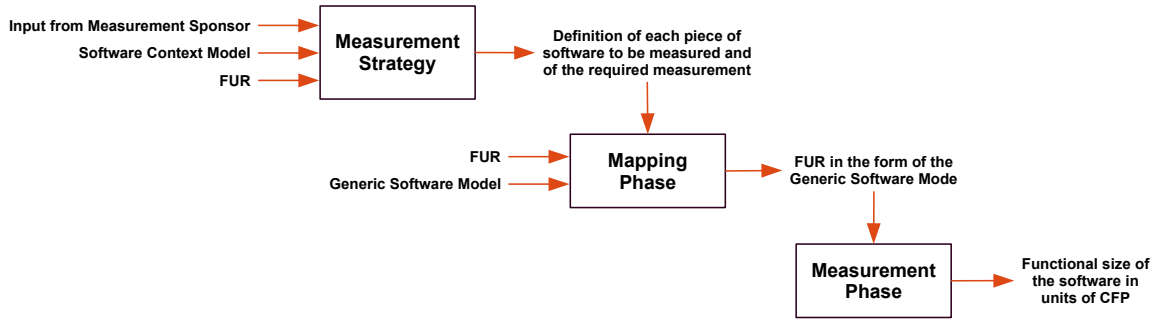
The weights for each unadjusted function points also use a similar method. Development people were asked about their effort distribution for the three parts of any transaction for a project i.e. effort distribution for *input*, *process* and *Output*. This is then used to estimate the weight for each of the components. Data collected form a single organization gives the organization average for the weights, and data from multiple organization can be used to calculate the industry average for the weights. Prior to the release of the MkII counting guideline in 1990, the average weights were the produced by the work done by *Nolan, Norton and Co.* on thirty-two systems covering a wide range of applications, environments and organisations. The industry average values finally set to $W_i = 0.58$,, $W_e = 1.66$ and $W_o = 0.26$. Intuitively, the performance gets better if the values are calibrated for the specific environment it is to be used. However, there is little incentive to carry out such a task. First of all, the impact of new technology is decreasing the utility and importance of TCA. Furthermore, the observed range of sizes of TCA is rather narrow (90% of TCA scores lie in the range 0.75 to 0.95, a factor of 1.27 in relative size). Thus, the importance of TCA

in computing the relative size of individual projects is minor, therefore, there is little to be gained in refining the TCA factor.

E.0.2 COSMIC Function Points

The ***C**ommon **S**oftware **M**easurement **C**onsortium (COSMIC)* is a later form of functional metric and after some years of research it was released in 1999. It was first developed as ***COSMIC–F**ull **F**unction **P**oint (COSMIC-FFP)*[3] in an attempt to meet the mandatory provisions of ISO/IEC 14141-1. It was also an attempt to bridge, what the people involved in COSMIC perceived as, the gap in the ability of existing methods to measure the size of real–time applications. Originally it was published as ISO/IEC 19761 COSMIC-FFP – A Functional Size Measurement Method. COSMIC has been promoted as a voluntary, world-wide grouping of software metrics experts in their official page. However, some of the original developers of the COSMIC function point have had long experience with functional metrics. Two of the most influential personalities were *Alain Abran*, who developed the ***F**ull **F**unction **P**oint (FFP)* (St Pierre et al., 1997) in Canada, and *Charles Symons*, who developed the *Mark II* (Symons, 1988) function point method in the United Kingdom. Features from both FFP and Mark II can be traced in the COSMIC method in the form of extended file and transaction types. COSMIC is a open and free–to–all measurement method and the guideline is freely available. The current version of the measurement guideline (i.e. ***M**easurement **M**anual (MM)*) is 4.0.1[4].

Fig. E.1. COSMIC Function Points Counting Process

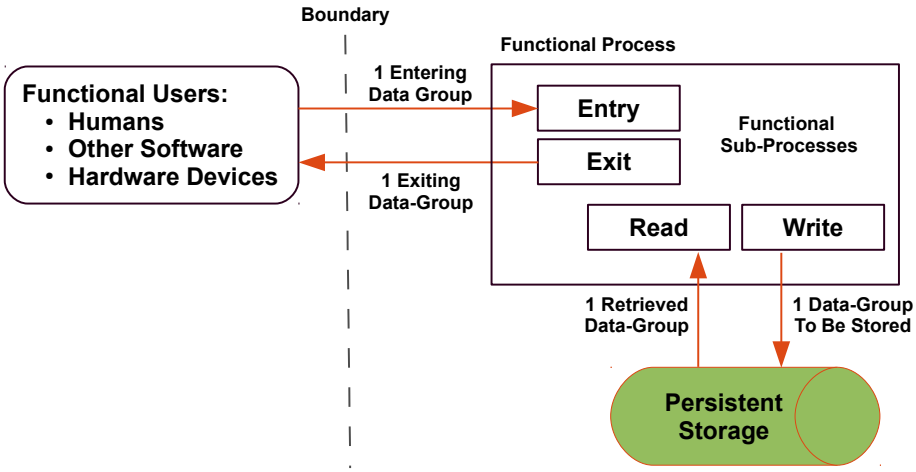


[3]The COSMIC management body changed the name from COSMIC–FFP to simply COSMIC and the measurement unit from ***COSMIC f**unctional **s**izing **u**nit (Cfsu)* to ***COSMIC F**unction **P**oint (CFP)* in 2007.

[4]http://www.cosmicon.com/portal/public/MMv4.0.1.pdf

Using COSMIC method one can measure the size of a software by defining the *purpose*, *scope* and *boundaries* of a measure at the beginning known as the *Software Context Model*. The ***F**unctional **U**ser **R**equirements (FUR)* is then collected and all these are used to produce FUR in the COSMIC generic software model during a phase called the *mapping phase*. Finally, all the identified components (i.e. FURs) are measured in terms of CFP and the result is then aggregated (see Figure E.1 for the visual presentation provided in COSMIC–MM v4.0.1). COSMIC measures the size of a software by measuring *four* distinct types of data movement characterizing the functional flow of data (see Figure E.2 for the visual presentation provided in COSMIC–MM v4.0.1).

Fig. E.2. COSMIC Function Points Measurement Process



The COSMIC principle suggests that the software gets input from a user and produced output required by the user. The manipulated information is data–groups and the functional size is proportional to the number of data movement necessary to perform the functional process. A data movement is a ***B**ase **F**unctional **C**omponent (BFC)* which moves a single data group type. There are *four* types of data movements,

◦ ***Entry*** is a data movement that moves data–group from a functional user across the application boundary into the functional process where it is required.
◦ ***Exit*** is a data movement that moves a data–group from a functional process across the boundary to the functional user that requires it.
◦ ***Read*** is a data movement that moves a data–group from persistent storage into the functional process which requires it.
◦ ***Write*** is a data movement that moves a data–group lying inside a functional process to persistent storage.

Single data–group involved in a data movement is a key features of the COSMIC measurement. The COSMIC manual explicitly states that data groups are mainly entities of an ER model in the third normal form. Thus, if more than one data–group is involved, each should be counted as a separate data movement element. The functional size in COSMIC solely depends on the number of data movement i.e. each detected data movement is defined as one CFP. Unlike IFPUG, there is no explicit measurement of files in the COSMIC method, only the access to files in terms of reading and writing operations.

Jones (1991) reported some experimental results comparing FP counts in IFPUG method to that of COSMIC and referred to the results to be somewhat *ambiguous*. He reported that for real–time and embedded systems the COSMIC count is often larger (between 15% and 50%) than that of IFPUG count. For, MIS he reported the results to be close with a variation of $\pm 10\%$. Other studies indeed show inconclusive results as well, for example, *The Fetcke study* (Fetcke, 1999) analysed four software applications of a data storage system and presented the correlation analysis. The sizes of the four projects were in *unadjusted IFPUG FP* (uFP) and *COSMIC CFP* and showed the relation $CFP = 1.1 \times uFP - 7.6$. Thus, using the conversion formula, a deviation of 4% in average (ranging from 0% to 8%) was found (Dumke and Abran, 2011).

E.0.3 The NESMA Function Points

The ***N**etherlands **S**oftware **M**etrics **A**ssociation (NESMA*[5]*)* released its first version of definition counting practice guideline in 1990, it assumed the principals of IFPUG counting practice (at that time the IFPUG–CPM was version 2.0). NESMA recognize *three* types of FP counting practices called, *detailed*, *estimated* and *indicative*. The detailed counting is almost parallel to the IFPUG method with some minor differences whereas the estimated method is a quicker version that uses constant values for the complexity ratings. The indicative measure however, is a very fast method and the most intriguing among the methods. The idea was to count FPs at a very early stage before all the details of the project is known and only the logical files are counted to determine an approximate FP count. The ILF count is multiplied by 35 and the EIF count is multiplied by 15 and the results are often comes within 15% of the count using the other two methods and also twice as fast as the detailed counting method (Jones, 1991). A detail description and comparative study between NESMA and IFPUG can be found in (Bundschuh and Dekkers, 2008, pp.389–393).

[5]http://nesma.org/

E.0.4 Other Variations

Capers Jones (Jones, 1991, pp. 106) presented a set of methods that are considered to be variants of the IFPUG-FSM method. Among the methods there were, prior versions of the IFPUG methos from v1.0 (1986) to 4.2 (2004), some well known but obsolete variations e.g. *Feature Points*, some fairly unknown and obsolete variant e.g. ***A**ustralian **S**oftware **M**etrics **A**ssociation (ASMA)* or Boeing's *3D Function Points*, successful variants e.g. NESMA and also some methods not necessarily based on IFPUG method e.g. *Object Points* (Bundschuh and Dekkers, 2008, pp. 397). Two of the methods classified as IFPUG variant with significantly interesting approach were *Feature Point* and *3D Function Points* and they will be addressed in this section due to their presence in relevant literature and historical significance.

One of the first modifications was proposed in 1986 by Capers Jones of ***S**oftware **P**roductivity **R**esearch (SPR)*. He published an experimental method based closely on that of Albrecht (1979), called *Feature Points* (Rahman, 2010), with the aim of extending FSM to apply to system software such as ***O**perating **S**ystem (OS)* or telephone switching systems and mathematically demanding algorithms. The method has been applied experimentally on OS, embedded systems, real–time systems, ***C**omputer **A**ided **D**esign (CAD)* applications, ***A**rtificial **I**ntelligence (AI)* systems and even ***M**anagement **I**nformation **S**ystem (MIS)* applications. He argued later (Jones, 2007) that it was a viable approach since FPA was originally developed to solve the measurement problems of classical MIS systems thus implying FPA's ineffectiveness in the aforementioned types of software due to their inherent algorithmic complexity. *Feature Point* method can been seen as a super set of FPA method and it was developed specifically to tackle high level of algorithmic and mathematical complexity. It introduces an *algorithm parameter* along with the original 5 FP parameters and reduced the average complexity weight of ILFs and EIFs to 7 from 10, thus implying the reduced significance of the logical files. However, the method has been largely abandoned due to the intrinsic difficulty of sizing mathematically rich algorithms.

In 1992 at Boeing Computer Services after 2 years of research, Scott A. Whitmire (1995) presented the *3D Function Points*, primarily to estimate size of real–time applications. The method made assumptions similar to that of the *Feature Point* but in contrast rather than contributing all the functional weight on the data and process, he proposed that control flow also contributes to the functional size. Thus, the term 3D represents the 3 dimensions of the method.

◦ ***Data dimension*** measured using the IFPUG-FSM
◦ ***Process dimension*** measured through data transformation i.e. considering the number of internal operations required to transform input to output data, and
◦ ***Control dimension*** measured using states and transitions i.e. counting the number of transitions between states.

Originally after counting all the dimensions, the values are added together to get the functional size. However, the details of the method is somewhat an industrial secret and it is also unclear if the method is still in practice or ever been used outside of Boeing. As we argued earlier, the presentation of the different counting practice demonstrate the existing diversity in the practice and gives a general idea of the difficulties involve in the automation process.

Appendix F

Results: Semi–Supervised Learning

F.1 Surface Form (`surface`)

Table F.1. Semi–Supervised Training Performance: $\texttt{surface}_{u}$*nigram*

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	9	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	4	9	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	4	9	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	4	9	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

Table F.2. Semi–Supervised Training Performance: `surface_bigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	11	9	1	16	1	0.900	0.9	0.900	0.900	0.941	0.941	0.926
BdF_DB	5	11	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852
BdF_MG	5	11	9	1	15	2	0.818	0.9	0.857	0.833	0.938	0.882	0.889
PSA	5	11	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852

Table F.3. Semi–Supervised Training Performance: `surface_trigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	15	9	2	15	1	0.900	0.818	0.857	0.882	0.882	0.938	0.889
BdF_DB	2	15	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	2	15	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	2	15	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.4. Semi–Supervised Training Performance: `surface_quardgram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	17	10	13	4	0	1	0.435	0.606	0.794	0.235	1	0.519
BdF_DB	4	17	6	17	4	0	1	0.261	0.414	0.638	0.190	1	0.370
BdF_MG	4	17	11	12	4	0	1	0.478	0.647	0.821	0.250	1	0.556
PSA	4	17	6	17	4	0	1	0.261	0.414	0.638	0.190	1	0.370

Table F.5. Semi–Supervised Training Performance: `surface_dgram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	17	9	2	15	1	0.900	0.818	0.857	0.882	0.882	0.938	0.889
BdF_DB	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	2	17	9	2	14	2	0.818	0.818	0.818	0.818	0.875	0.875	0.852
PSA	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.6. Semi–Supervised Training Performance: `surface_dgram_wild`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	11	9	0	17	1	0.900	1.000	0.947	0.918	1.000	0.944	0.963
BdF_DB	5	11	6	3	18	0	1.000	0.667	0.800	0.909	0.857	1.000	0.889
BdF_MG	5	11	9	0	16	2	0.818	1.000	0.900	0.849	1.000	0.889	0.926
PSA	5	11	6	3	18	0	1.000	0.667	0.800	0.909	0.857	1.000	0.889

Table F.7. Semi–Supervised Training Performance: `surface_extended_dgram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	17	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	3	17	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	3	17	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	3	17	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

Table F.8. Semi–Supervised Training Performance: surface_dep_subtree

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	15	10	2	15	0	1	0.833	0.909	0.962	0.882	1	0.926
BdF_DB	5	15	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778
BdF_MG	5	15	11	1	15	0	1	0.917	0.957	0.982	0.938	1	0.963
PSA	5	15	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778

F.2 Lemma (`lemma`)

Table F.9. Semi–Supervised Training Performance: `lemma_unigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	9	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	3	9	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	3	9	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	3	9	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.10. Semi–Supervised Training Performance: `lemma_bigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	13	9	1	16	1	0.900	0.9	0.900	0.900	0.941	0.941	0.926
BdF_DB	4	13	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852
BdF_MG	4	13	10	0	16	1	0.909	1.0	0.952	0.926	1.000	0.941	0.963
PSA	4	13	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852

Table F.11. Semi–Supervised Training Performance: `lemma_trigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	1	11	8	3	14	2	0.800	0.727	0.762	0.784	0.824	0.875	0.815
BdF_DB	1	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	1	11	9	2	14	2	0.818	0.818	0.818	0.818	0.875	0.875	0.852
PSA	1	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.12. Semi–Supervised Training Performance: lemma_quardgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	15	7	1	16	3	0.700	0.875	0.778	0.729	0.941	0.842	0.852
BdF_DB	5	15	4	4	17	2	0.667	0.500	0.571	0.625	0.810	0.895	0.778
BdF_MG	5	15	8	0	16	3	0.727	1.000	0.842	0.769	1.000	0.842	0.889
PSA	5	15	4	4	17	2	0.667	0.500	0.571	0.625	0.810	0.895	0.778

Table F.13. Semi–Supervised Training Performance: lemma_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	17	10	4	13	0	1.000	0.714	0.833	0.926	0.765	1.000	0.852
BdF_DB	3	17	6	8	13	0	1.000	0.429	0.600	0.789	0.619	1.000	0.704
BdF_MG	3	17	10	4	12	1	0.909	0.714	0.800	0.862	0.750	0.923	0.815
PSA	3	17	6	8	13	0	1.000	0.429	0.600	0.789	0.619	1.000	0.704

Table F.14. Semi–Supervised Training Performance: lemma_dgram_wild

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	15	9	3	14	1	0.900	0.750	0.818	0.865	0.824	0.933	0.852
BdF_DB	5	15	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778
BdF_MG	5	15	10	2	14	1	0.909	0.833	0.870	0.893	0.875	0.933	0.889
PSA	5	15	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778

Table F.15. Semi–Supervised Training Performance: lemma_extended_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	17	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	2	17	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.16. Semi–Supervised Training Performance: lemma_dep_subtree

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	11	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	4	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	4	11	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	4	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

F.3 General POS (gpos)

Table F.17. Semi–Supervised Training Performance: gpos_unigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	17	10	2	15	0	1	0.833	0.909	0.962	0.882	1	0.926
BdF_DB	5	17	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778
BdF_MG	5	17	11	1	15	0	1	0.917	0.957	0.982	0.938	1	0.963
PSA	5	17	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778

Table F.18. Semi–Supervised Training Performance: gpos_bigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	17	10	3	14	0	1	0.769	0.870	0.943	0.824	1	0.889
BdF_DB	2	17	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741
BdF_MG	2	17	11	2	14	0	1	0.846	0.917	0.965	0.875	1	0.926
PSA	2	17	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741

Table F.19. Semi–Supervised Training Performance: gpos_trigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	13	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	3	13	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	3	13	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	3	13	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

Table F.20. Semi–Supervised Training Performance: gpos_quardgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	11	9	1	16	1	0.900	0.9	0.900	0.900	0.941	0.941	0.926
BdF_DB	3	11	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852
BdF_MG	3	11	10	0	16	1	0.909	1.0	0.952	0.926	1.000	0.941	0.963
PSA	3	11	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852

Table F.21. Semi–Supervised Training Performance: gpos_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	17	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	5	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	5	17	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	5	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.22. Semi–Supervised Training Performance: gpos_dgram_wild

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	15	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	5	15	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	5	15	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	5	15	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

Table F.23. Semi–Supervised Training Performance: gpos_extended_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	17	10	4	13	0	1	0.714	0.833	0.926	0.765	1	0.852
BdF_DB	2	17	6	8	13	0	1	0.429	0.600	0.789	0.619	1	0.704
BdF_MG	2	17	11	3	13	0	1	0.786	0.880	0.948	0.813	1	0.889
PSA	2	17	6	8	13	0	1	0.429	0.600	0.789	0.619	1	0.704

Table F.24. Semi–Supervised Training Performance: gpos_dep_subtree

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	17	10	4	13	0	1	0.714	0.833	0.926	0.765	1	0.852
BdF_DB	4	17	6	8	13	0	1	0.429	0.600	0.789	0.619	1	0.704
BdF_MG	4	17	11	3	13	0	1	0.786	0.880	0.948	0.813	1	0.889
PSA	4	17	6	8	13	0	1	0.429	0.600	0.789	0.619	1	0.704

F.4 Specific POS (spos)

Table F.25. Semi–Supervised Training Performance: spos_unigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	13	9	1	16	1	0.900	0.9	0.900	0.900	0.941	0.941	0.926
BdF_DB	2	13	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852
BdF_MG	2	13	10	0	16	1	0.909	1.0	0.952	0.926	1.000	0.941	0.963
PSA	2	13	6	4	17	0	1.000	0.6	0.750	0.882	0.810	1.000	0.852

Table F.26. Semi–Supervised Training Performance: spos_bigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	9	10	2	15	0	1.000	0.833	0.909	0.962	0.882	1.000	0.926
BdF_DB	5	9	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778
BdF_MG	5	9	10	2	14	1	0.909	0.833	0.870	0.893	0.875	0.933	0.889
PSA	5	9	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778

Table F.27. Semi–Supervised Training Performance: spos_trigram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	17	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	3	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	3	17	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	3	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.28. Semi–Supervised Training Performance: spos_quardgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	13	10	2	15	0	1	0.833	0.909	0.962	0.882	1	0.926
BdF_DB	3	13	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778
BdF_MG	3	13	11	1	15	0	1	0.917	0.957	0.982	0.938	1	0.963
PSA	3	13	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778

Table F.29. Semi–Supervised Training Performance: spos_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	11	10	1	16	0	1.000	0.909	0.952	0.980	0.941	1.000	0.963
BdF_DB	5	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	5	11	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	5	11	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.30. Semi–Supervised Training Performance: spos_dgram_wild

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	17	9	2	15	1	0.900	0.818	0.857	0.882	0.882	0.938	0.889
BdF_DB	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	2	17	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	2	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

Table F.31. Semi–Supervised Training Performance: spos_extended_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	17	10	2	15	0	1.000	0.833	0.909	0.962	0.882	1.000	0.926
BdF_DB	5	17	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778
BdF_MG	5	17	10	2	14	1	0.909	0.833	0.870	0.893	0.875	0.933	0.889
PSA	5	17	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778

Table F.32. Semi–Supervised Training Performance: spos_dep_subtree

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	17	9	2	15	1	0.900	0.818	0.857	0.882	0.882	0.938	0.889
BdF_DB	3	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815
BdF_MG	3	17	10	1	15	1	0.909	0.909	0.909	0.909	0.938	0.938	0.926
PSA	3	17	6	5	16	0	1.000	0.545	0.706	0.857	0.762	1.000	0.815

F.5 Lemma & General POS Together (`lemma_gpos`)

Table F.33. Semi–Supervised Training Performance: `lemma_gpos_unigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	15	10	0	17	0	1.000	1.0	1.000	1.000	1.00	1.000	1.000
BdF_DB	4	15	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852
BdF_MG	4	15	10	0	16	1	0.909	1.0	0.952	0.926	1.00	0.941	0.963
PSA	4	15	6	4	17	0	1.000	0.6	0.750	0.882	0.81	1.000	0.852

Table F.34. Semi–Supervised Training Performance: `lemma_gpos_bigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	11	10	2	15	0	1	0.833	0.909	0.962	0.882	1	0.926
BdF_DB	4	11	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778
BdF_MG	4	11	11	1	15	0	1	0.917	0.957	0.982	0.938	1	0.963
PSA	4	11	6	6	15	0	1	0.500	0.667	0.833	0.714	1	0.778

Table F.35. Semi–Supervised Training Performance: `lemma_gpos_trigram`

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	13	10	7	10	0	1	0.588	0.741	0.877	0.588	1	0.741
BdF_DB	5	13	6	11	10	0	1	0.353	0.522	0.732	0.476	1	0.593
BdF_MG	5	13	11	6	10	0	1	0.647	0.786	0.902	0.625	1	0.778
PSA	5	13	6	11	10	0	1	0.353	0.522	0.732	0.476	1	0.593

Table F.36. Semi–Supervised Training Performance: lemma_gpos_quardgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	11	10	15	2	0	1	0.40	0.571	0.769	0.118	1	0.444
BdF_DB	2	11	6	19	2	0	1	0.24	0.387	0.612	0.095	1	0.296
BdF_MG	2	11	11	14	2	0	1	0.44	0.611	0.797	0.125	1	0.481
PSA	2	11	6	19	2	0	1	0.24	0.387	0.612	0.095	1	0.296

Table F.37. Semi–Supervised Training Performance: lemma_gpos_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	3	17	10	3	14	0	1	0.769	0.870	0.943	0.824	1	0.889
BdF_DB	3	17	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741
BdF_MG	3	17	11	2	14	0	1	0.846	0.917	0.965	0.875	1	0.926
PSA	3	17	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741

Table F.38. Semi–Supervised Training Performance: lemma_gpos_dgram_wild

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	5	17	9	3	14	1	0.900	0.750	0.818	0.865	0.824	0.933	0.852
BdF_DB	5	17	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778
BdF_MG	5	17	10	2	14	1	0.909	0.833	0.870	0.893	0.875	0.933	0.889
PSA	5	17	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778

Table F.39. Semi–Supervised Training Performance: lemma_gpos_extended_dgram

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	2	15	10	2	15	0	1.000	0.833	0.909	0.962	0.882	1.000	0.926
BdF_DB	2	15	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778
BdF_MG	2	15	10	2	14	1	0.909	0.833	0.870	0.893	0.875	0.933	0.889
PSA	2	15	6	6	15	0	1.000	0.500	0.667	0.833	0.714	1.000	0.778

Table F.40. Semi–Supervised Training Performance: lemma_gpos_dep_subtree

Annotator	PC	K	TP	FP	TN	FN	R	P	F1	F2	SPC	NPV	ACC
Acapi_JNV	4	13	10	3	14	0	1	0.769	0.870	0.943	0.824	1	0.889
BdF_DB	4	13	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741
BdF_MG	4	13	11	2	14	0	1	0.846	0.917	0.965	0.875	1	0.926
PSA	4	13	6	7	14	0	1	0.462	0.632	0.811	0.667	1	0.741

Appendix G

The History of MUCs

G.1 MUC–1

It was held in May, 1987 in San Diego with six systems participated for the task. There was neither a predefined output format nor a formal evaluation. Therefore, it was not possible to compare one system with other in terms of performance. The development corpus consisted of 10 Navy ***Op****erational* ***Rep****ort (OPREP)* messages on ship sightings and engagements, each of them was just a few sentences long. Each participant proposed a suitable output for each message and two unseen messages were distributed at the conference for testing.

G.2 MUC–2

It was held in May, 1989 with eight systems participated for the task. The output format was defined as a template with 10 slots for attributes to fill and the data came from the same domain as MUC–1. Resources in the form of lists of specialised naval terminology were also supplied. Having a shared output format allowed the comparison of individual systems. The participants would receive a description of class of events to be identified in the source text. The template has slots for information about the event, such as the type of event, the agent, the time and place, the effect, etc. 105 messages were supplied as training data and there were two test rounds. The first round with 20 blind messages and then, after system fixes, a second round of 5 blind messages just before the conference. There was a standardized primary evaluation metric developed during MUC–2 as well. Evaluation criteria were defined, but by consensus deemed not to have been adequate (Gaizauskas and Wilks, 1997).

G.3 MUC–3

It was held in May 1991 in San Diego with 15 systems participated. This time the source texts were shifted to news articles about terrorist activities in Central and South America. The stories came from an electronic database, although collected originally from different news media (e.g. newspaper, television, radio, speech, interviews etc.) . Since most of them were in Spanish, many of them were translated by the *US Foreign Broadcast Information Service*. The development set had 1,300 text and 3 test sets of 100 texts were supplied. Template complexity was increased by using a template of 18 slots. During MUC–3 formal evaluation criteria ware established that have been adapted from the notions of formal evaluation of IR. Although, official scoring was performed by the organizers, a semi–automated scoring program was developed and made available for use by participants during development.

G.4 MUC–4

The TIPSTER Text Program is a DERPA led initiative towards advancing the state–of–the–art in text processing, namely IR and IE through the cooperation of the researchers and developers in the government (US), industry and academia. The real contribution to IE is the creation of common software architecture and industry standard for multi–component IE system. Some of the influential systems that followed this architecture are,

- ***C**enter for **R**esearch **L**ibraries (CRL)* Temple MT System (Vanni and Zajac, 1996)
- OLEADA language training system (Ogden and Bernick, 1996)
- Sheffield GATE[a] system (Cunningham et al., 1995)

[a] http://gate.ac.uk/

MUC–4 was held in June, 1992 in McLean, Virginia with seventeen participants The dataset domain and the template structure was essentially remained the same as that of MUC–3. Task definition, corpus, performance metric and test protocol was altered and improved significantly focusing on generating negative data points to assess the systems' independence from the training data. The alterations were focusing on generating negative data points to assess the systems' independence from the training data. Thus scoring of MUC–4 is considered to be more consistent and reliable comparison between system performances.

It was also the beginning of the MUC conferences' inclusion within the *TIPSTER Text Program*[1]

G.5 MUC–5

It was held in August 1993 in Baltimore, Maryland[2] with 17 participants. Of them 14 American, 1 Canadian, 1 British and 1 Japanese participant, thus marking the first international involvement in the MUC's. Moreover, evaluation was performed in 2 languages, English and Japanese, significant amount of additional resources were supplied, test corpora size was increased, scoring was modified to include new metrics and the scoring program was enhanced significantly. Similar to MUC–3 and MUC–4 news wire stories were used but from two different domains, Joint Ventures (abbreviated JV) and Microelectronics (ME). The domain–language pairs were named EJV, JJV, EME, JME, where the first character is the language reference (E for English and J for Japanese in a very obvious manner). *Non–TIPSTER–sponsored* systems had to choose at lest 1 domain and one language whereas, TIPSTER–sponsored systems were suppose to operate in all domain–language pairs. However most end up choosing only one task proving the challenging nature and EJV was the most popular and by common consent the most difficult.

There were 3 sources for the EJV material: called the *Wall Street Journal*, *LEXUS/NEXUS* and *PROMT* collected from from ACL/DCI or TIPSTER Detection database CDROMs by using traditional keyword–based (keywords for EJV included such stems as joint venture, joint, venture, tie–up, collaborate, cooperate etc.) document retrieval systems. Filled–out templates for approximately 1000 documents of each training set were provided as *keys*. In addition, templates were produced for the initial TIPSTER program test cycles (12 and 18 months) and for the final joint MUC–5/TIPSTER (24 month) test (Carlson et al., 1993).

The MUC–5 template and fill rules are considered to be the most complex to date due to the presence of nested data structure i.e. a slot was allowed to have a pointer to another slot. Therefore, the template has a object–oriented feel to it, e.g. a joint–venture was viewed as an object with slots like name, status (e.g. existing, dissolved etc.) but also had slots for the participating organizations in the venture, each of which were pointers to an organization object, containing slots themselves and sometimes may even have pointer to some other

[1] http://www.itl.nist.gov/iaui/894.02/related_projects/tipster/overv.htm
[2] coinciding with the TIPSTER–I 24-month evaluation

object. In all there were 11 object and 49 slots to be filled. As an indication of the level of detail required to define the extraction task, the fill rules occupied a 45 page document.

During the development of the templates and the fill–rules, several changes had been made at different stages, thus, every modification implies make an update to all the previously instantiated templates. As an indication of the extent of the process Gaizauskas and Wilks (1997) presented that The cost of producing the answer keys alone for MUC-5 and for the preceding TIPSTER extraction trials was more than $1 million US. They (Gaizauskas and Wilks, 1997) also pointed an interesting observation that in each domain the Japanese scores were higher and thus prompted discussion of whether in some sense Japanese is an easier language from which to extract information. The evaluation metric had some additional raw measures along with standard precision and recall, namely, error per response fill, under–generation, over–generation, and substitution. In contrast to precision and recall a lower score is better for these measures.

G.6 MUC–6

Organized in 1995, MUC–6 is the first MUC, reminiscence of the modern framework of IE. Instead of a single all–in–one system evaluation like MUC–5, participants were offered a choice of smaller subtasks, often referred to as *mini MUCs*. This templates resemble MUC–2 like templates rather than MUC–5 because of the portability reason but the nested architecture from MUC–5 was kept. The restructuring was the direct result of the interest and intended focus of different participants for specific tasks and the fact that all–in–one IE system was getting too large to accommodate the ever growing task list under a single framework. As a matter of fact the growth was not only a research issue but also accompanying engineering issues. In MUC–6 there were four task definitions: along with the classic template filling, three new tasks were introduced,

1. **Named Entity Task (NE)**: Identifying constituents in the text, which represents a person, an organization, a location name, a date, a currency etc.

```
Mr. <ENAMEX TYPE=“PERSON”> Dooner </ENAMEX> met with <ENAMEX TYPE = “PERSON”>
Martin Puris </ENAMEX>, president and chief executive officer of <ENAME XTYPE
= “ORGANIZATION”> Ammirati & Puris </ENAMEX>, about <ENAME XTYPE = “ORGANIZA
TION”> McCann </ENAMEX>’s acquiring the agency with billings of <NUME XTYPE =
“MONEY”> $400 million </NUMEX>.
```

2. **Template Element Task (TE)**: Identifying descriptions of entities i.e. identifying individual template elements.

```
<ORGANIZATION-9402240133-5> :  _
   ORG_NAME : "Coca-Cola"
   ORG_ALIAS : "Coke"
   ORG_TYPE : COMPANY
   ORG_LOCALE : Atlanta CITY
   ORG_COUNTRY : United States
```

3. **Co-reference Task (CO)**: Identifying co-referring constituents, thus all mentions of a given entity and it used identified constituents from NE and TE tasks.

```
Maybe <COREF ID="136" REF="134"> he </COREF>'ll even leave something from
<COREF ID="138" REF="139"><COREF ID="137"" REF="136"> his </COREF> office
</COREF> for <COREF ID="140" REF="91"> Mr . Dooner </COREF>.  Perhaps <COREF
ID="144"> a framed page from the New York Times, dated Dec. 8, 1987, showing
a year-end chart of the stock market crash earlier that year</COREF>.  <COREF
ID="141" REF="137"> Mr . James </COREF> says <COREF ID="142" REF="141"> he
</COREF> framed <COREF ID="143" REF="144" STATUS="OPT"> it </COREF> and kept
<COREF ID="145" REF="144"> it </COREF> by <COREF ID="146" REF="142"> his
</COREF> desk as a personal reminder.  It can all be gone like that.
```

All the examples above is taken from (Grishman and Sundheim, 1995). The primary resources were supplied by the organizer in the form of development and test corpora and scoring software, for both the dry run and final evaluation. There were 100 annotated texts for each of the four tasks in the development corpus and there were 30 annotated texts for the NE and CO tasks and 100 texts for TE and the classic template filling tasks. Texts were from *Wall Street Journal* text and new scoring software was developed for NE and CO tasks whereas, the MUC–5 scoring software was enhanced for the template tasks. The evaluation reverted to precision and recall from the error–per–response–fill metric used in MUC–5. The template tasks were scored as in MUC–5 with modifications and new measurement metric had been developed for the NE and CO tasks in terms of precision and recall. A combined F-Measure was the decisive ranking for NE but no such measure was possible for the CO task because of the varying precision and recall measures for the task.

G.7 MUC–7

It was organized in 1997 with 18 participants and the data came from *New York Times News Service* supplied by *Linguistic Data Consortium (LDC)*[3]. The target domains were, aircraft accident and launch events and relevant documents were extracted from approximately 158,000 articles using *Managing Gigabytes* text retrieval system. There were 2 sets of 100 articles (aircraft accident domain) primarily training, including dry run and 2 sets of 100 articles selected balanced for relevancy, type and source for formal run (launch event domain). The task definitions for MUC-7 were improved by having authors other than the original authors revise each of the guidelines for internal consistency and to dovetail into the other tasks evaluated. The communal effort in polishing the guidelines and the data mark–up noticeably improved the evaluation. MUC-7 included two new tasks,

1. **Multi–lingual Entity Task (MEN)**: NE task for Chinese and Japanese.
2. **Template Relation Task (TR)**: Identifying relational information between entities (e.g. employee_of, manufacture_of, and location_of relations).

[3]http://projects.ldc.upenn.edu