Ingénierie agile de lignes de produits logiciels pour des applications d'aide à la décision pour l'agriculture

Thomas Georges

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale Information, Structures, Systèmes

Unité de recherche LIRMM

Ingénierie agile de lignes de produits logiciels pour des applications d'aide à la décision pour l'agriculture

Présentée par Thomas GEORGES
le 23 janvier 2024

Sous la direction de Chouki TIBERMACINE

Devant le jury composé de

Peggy CELLIER, Maître de conférence, IRISA, INSA Rennes, Université de Rennes, France	**Rapportrice**
Gilles PERROUIN, Chercheur qualifié, FNRS, NADI-PReCISE, Université de Namur, Belgique	**Rapporteur**
Sophie EBERSOLD, Professeure, SM@RT, IRIT, Université de Toulouse, France	**Examinatrice**
Raul MAZO, Professeur, LabSTICC, ENSTA Bretagne, France	**Examinateur**
Vincent BERRY, Professeur, LIRMM, Université de Montpellier, France	**Président du jury**
Marianne HUCHARD, Professeure, LIRMM, Université de Montpellier, France	**Co-encadrante**
Clémentine NEBUT, Maître de conférence, LIRMM, Université de Montpellier, France	**Co-encadrante**
Mélanie KÖNIG, Docteure, Entreprise ITK, France	**Invitée**

UNIVERSITÉ DE MONTPELLIER

Résumé

L'industrialisation du développement des logiciels en entreprise favorise la création de solutions logicielles sur mesure, de meilleure qualité et à moindre coût. Dans ce contexte, la migration vers une ligne de produits logiciels (LPL) représente une option intéressante pour les entreprises qui proposent déjà une gamme de produits logiciels similaires. Le partenaire industriel de ces travaux de recherche est l'entreprise ITK. Cette entreprise développe une gamme de produits logiciels d'aide à la décision pour différentes cultures agricoles et a pour projet de migrer vers une LPL. Afin de favoriser une migration réussie vers une ligne de produits logiciels, il est essentiel de préparer les différents acteurs du projet à adopter de nouvelles méthodes et pratiques. C'est pourquoi une évaluation approfondie des attentes et de la préparation des parties prenantes est nécessaire, permettant ainsi de faciliter cette transition. La première contribution de cette thèse consiste à réaliser cette évaluation. Elle a été réalisée au moyen d'entretiens avec les acteurs du projet et d'une analyse approfondie de leurs réponses. La migration de la gamme de logiciels existants vers une LPL implique de localiser et d'identifier les caractéristiques existantes. La création de modèles de caractéristiques représentatifs du système à migrer constitue une première étape cruciale. De nombreuses ressources sont liées au code source et peuvent être exploitées pour enrichir la migration, la génération de modèles et la maintenance de la LPL. Les plateformes de gestion de versions du code source offrent par exemple des moyens de lier les spécifications au code source, notamment grâce à l'utilisation *récits utilisateurs* (*user stories*) et de *fusions* (*merge requests*) de code par caractéristiques. La deuxième contribution de cette thèse est un processus automatisé qui utilise les récits utilisateurs pour générer des modèles de caractéristiques. Cette méthode combine le traitement du langage naturel, des modèles supervisés d'intelligence artificielle et l'analyse formelle et relationnelle de concepts afin de générer des modèles de caractéristiques pour chaque utilisateur des logiciels étudiés. Ces modèles sont enrichis en tenant compte des contraintes issues des fusions de code et d'une ontologie du domaine. La troisième contribution de cette thèse est une méthode identifiant la variabilité dans les schémas de données des différents simulateurs des logiciels existants. En effet, le développement d'un nouveau logiciel chez l'entreprise ITK débute d'abord par l'intégration d'un simulateur, développé par les agronomes, pour une nouvelle culture. Cette méthode est basée sur l'analyse formelle de concepts et permet d'enrichir les configurations pour de créer de nouveaux produits.

Remerciements

Je souhaite débuter mes remerciements en exprimant ma profonde gratitude envers mon directeur de thèse, Chouki Tibermacine, ainsi qu'à mes co-encadrants exceptionnels, Marianne Huchard, Clémentine Nebut, et Mélanie König. Leur bienveillance, leur expertise et leurs conseils ont grandement contribué à la réussite de ces travaux de recherche.

Je tiens à adresser mes sincères remerciements aux rapporteurs, Peggy Cellier et Gilles Perrouin, pour avoir pris le temps de lire ma thèse et pour leurs remarques. Un grand merci également aux membres du jury, en particulier à Vincent Berry, président du jury, Sophie Ebersold et Raul Mazo Peña, pour leur présence lors de ma soutenance.

Ce projet n'aurait pas pu voir le jour sans la précieuse collaboration de l'entreprise ITK, qui m'a accueilli et m'a offert une liberté totale pour mes recherches sur leur famille de logiciels. Mes sincères remerciements à tous ceux avec qui j'ai eu le privilège de collaborer, en particulier les membres de l'équipe Spacecrop et de l'ensemble d'ITK. Je tiens également à remercier spécialement Mélanie König et Loïc Bourgeois pour leur encadrement du côté de l'entreprise.

L'environnement stimulant du LIRMM a eu un impact majeur sur ma réussite. Je souhaite donc exprimer ma gratitude à tous les chercheurs du LIRMM qui m'ont accompagné tout au long de ma thèse. Un merci particulier à Eric Bourreau, Mathieu Lafourcade et Christophe Fiorio pour leurs conseils et leur présence lors de mes comités de suivi. Je tiens également à remercier les enseignants du département informatique de la faculté des sciences de Montpellier et de Polytech.

Je souhaite adresser mes remerciements aux étudiants que j'ai rencontrés lors de mes missions d'enseignement, ainsi qu'à ceux que j'ai encadrés en stage et en projet.

Un grand merci à mes collègues doctorants, qui ont contribué à créer une ambiance positive. Merci à Bachar, Bianca, Camille, Florent, Nicolas, Pascal, Séléna, Sophie et Vincent. Je tiens à exprimer ma gratitude envers les amis rencontrés lors de l'événement "Ma thèse

en 180 secondes" avec qui j'ai toujours plaisir à recroiser : Caroline, Flo, Hugo, Ines, Julie, Lucie, Mathilde, Maëlys, Nicolas, Noémie, Pauline, Priscillia et Théo.

Je souhaite également remercier mes amis qui m'ont soutenu tout au long de ces trois années : Armandine, Alexis, Clément, Damien, Elie, Justine, Margot, Matthieu et Loïc. Des remerciements spéciaux vont à Luc et Simon, qui m'accompagnent depuis mes débuts dans l'informatique il y a 8 ans.

Un merci tout particulier à Laurie, qui a toujours été présente pour me soutenir et m'encourager. Ta constante inspiration et ton soutien inébranlable ont été inestimables dans les moments difficiles comme dans les moments heureux.

"Enfin, un dernier remerciement à Julie, consoeur infatigable de thèse, sans qui les moments passés au LIRMM et à l'université, que ce soit en tant qu'étudiant ou en enseignement, n'auraient pas été aussi plaisants.

Table des matières

CHAPITRE 1

Introduction

Sommaire

Pour introduire cette thèse, il est essentiel de se familiariser avec le contexte, la problématique et les objectifs sous-jacents de ce projet de recherche. Les travaux présentés résultent d'une collaboration fructueuse avec l'entreprise ITK, qui a développé une famille de logiciels. Notre étude s'est concentrée sur la manière d'encourager et de soutenir la migration de l'existant vers l'ingénierie des lignes de produits (LPL) au sein de cette famille de logiciels.

Cette introduction a pour dessein de jeter la lumière sur le contexte à l'origine de ce projet de recherche et de mettre en évidence les questions qui l'ont façonné. En procédant ainsi, nous établissons les fondations du projet et exposons les problématiques qui seront abordées.

1.1 Contexte

Dans cette partie sont présentés les éléments du contexte qui serviront à appréhender nos problématiques liées aux lignes de produits logiciels (LPL) et au développement agile de logiciels. Ces éléments prennent en compte les pratiques chez notre partenaire ITK. Chacun de ces éléments contribue à la compréhension des enjeux de notre projet de recherche.

1.1.1 Gamme de produits logiciels

Les LPL sont désormais une manière bien établie de produire efficacement des produits logiciels hautement configurables [PBvdL05]. Une ligne de produits logiciels est une infrastructure logicielle comprenant entre autres une architecture, des modèles de variabilité et des actifs (assets), tout en favorisant la personnalisation des logiciels au sein d'une famille. Les modèles de variabilité exposent les caractéristiques de produits disponibles et les manières autorisées de les combiner dans des configurations de produits. Une base d'actifs *(assets)* fournit les morceaux de logiciels concrets qui implémentent les caractéristiques. Les outils servant dans l'ingénierie des LPL permettent la sélection des caractéristiques pour créer une configuration concrète, puis génèrent le produit concret à partir de la configuration en assemblant les actifs de la base. Une telle approche devient intéressante lorsqu'un nombre significatif de configurations doivent être développées. Cependant, cela n'est pas évident à planifier pour une entreprise qui débute avec un nouveau type de produits. Ainsi, de nombreuses entreprises développent d'abord une famille de systèmes logiciels similaires avant de décider d'investir des efforts dans la migration de leur infrastructure de développement logiciel vers une ligne de produits.

Malgré les expériences réussies rapportées dans la littérature [AJL+20], la migration d'une famille de logiciels existante vers une plateforme LPL intégrée reste un défi et les organisations peuvent hésiter à adopter cette approche. Les hésitations proviennent souvent du manque de procédures standards guidant le processus, ainsi que des doutes sur le rapport coût/bénéfice. Ainsi, bien que les entreprises possèdent de vastes bases de code bien documentées, elles continuent à construire manuellement des applications sur mesure à partir du code de base pour répondre aux besoins de leurs clients, en particulier avec une stratégie de clonage et de personnalisation ad hoc.

1.1.2 Développement agile de logiciels

L'agilité représente un ensemble de principes et de pratiques qui mettent en avant l'amélioration continue, la proximité avec les clients et la prise de décisions collectives. Dans le contexte du développement logiciel, elle englobe un ensemble de méthodes et de pratiques visant à produire du code de meilleure qualité de manière efficace et rapide. Les fondements et les valeurs de ces pratiques sont documentés dans un ouvrage clé, le

Manifeste Agile [BBvB+01]. Celui-ci promeut la livraison continue de logiciels fonctionnels au client plutôt que d'attendre un temps plus long et que le logiciel soit totalement achevé. Cette approche favorise la collaboration tant au sein des équipes qu'avec les clients, ce qui permet de prévenir l'effet tunnel et ses problèmes potentiels.

1.1.3 L'entreprise ITK - Predict and Decide

Ce travail de recherche a été mené avec un partenaire industriel, ITK - Predict and Decide [1], qui est une PME comptant 120 collaborateurs au moment de la rédaction du mémoire, dont 19 agronomes et 29 développeurs, 2 concepteurs UI/UX et 4 administrateurs système. Les applications développées aident les agriculteurs dans la prise de décision en leur permettant de prédire les maladies et les rendements des cultures pour leurs plantes et leur bétail. Ces applications sont construites sur des simulateurs développés par des agronomes. Ensuite, les équipes informatiques construisent le code autour des simulateurs. Ce code collecte des données à partir de dispositifs de type Internet des objets (IoT) ou de services en ligne afin d'alimenter les simulateurs et d'afficher leurs données de sortie dans des tableaux de bord.

Nous avons étudié une gamme de produits logiciels pour différents types de cultures, qu'elles soient pérennes ou annuelles. Elle est spécialisée dans les analyses pour mesurer et améliorer la rentabilité, en se concentrant sur l'optimisation de la performance de production et de la chaîne logistique.

L'implémentation de la gamme de produits est faite sur une plate-forme contenant tout le code source avec une séparation entre ce qui est commun et ce qui est spécifique aux produits. Les produits sont des applications web. Le serveur et les services sont implémentés en Kotlin, avec Spring Boot. Les clients sont implémentés en Typescript, avec Angular. Ce sont des langages et des technologies couramment utilisés notamment dans les approches de lignes de produits logiciels. Les langages orientés objets sont aussi particulièrement enclins à la réutilisabilité.

Les prévisions, conseils et risques sont fournis par des simulateurs développés par les agronomes d'ITK. Chaque simulateur peut être vu comme une boîte noire avec des entrées et des sorties. Par exemple le simulateur de maladies prendra en entrée le stade phénologique de la plante, sa localisation et la météo pour fournir en sortie un pourcentage de risque et la période à risque pour différentes maladies.

L'ensemble des pratiques au sein des équipes sont issues de l'agilité pour un fonctionnement géré au sein des équipes par les acteurs du projet eux-mêmes.

La proximité entre les processus au sein d'ITK et de l'ingénierie des LPL nous a orientés vers plusieurs problématiques et pistes de recherches que nous avons étudiées.

1. www.itk.fr

1.2 Problématique

Cette thèse aborde les besoins et les problématiques qui en découlent, chacune contribuant à une compréhension approfondie des objectifs.

La première est l'élaboration d'un cadre pour obtenir et analyser les perceptions des acteurs en vue d'une migration, ainsi que pour guider cette migration :

- **Comment les acteurs perçoivent-ils les bonnes pratiques de développement actuelles ?** Cette question vise à mettre en évidence les habitudes des acteurs en matière de développement actuel, en vue d'une comparaison avec les nouvelles pratiques post-migration.
- **Quelle est la vision des acteurs de l'entreprise quant aux avantages potentiels de la future LPL, en particulier en ce qui concerne le développement d'applications ?** Cette interrogation explore les attentes et les espoirs des acteurs vis-à-vis de la LPL et de son impact sur le développement d'applications.
- **Quelle est l'opinion des acteurs de l'entreprise sur les risques associés à la future LPL et comment perçoivent-ils son impact sur leur travail quotidien ?** Cette question vise à comprendre la manière dont les acteurs perçoivent les défis et les risques associés à la migration vers la LPL, ainsi que leur ressenti quant à l'impact sur leurs tâches quotidiennes.
- **Quelles sont les craintes spécifiques des acteurs de l'entreprise liées aux risques identifiés précédemment, afin de mieux appréhender les conséquences potentielles du projet de migration ?** Cette interrogation permet d'explorer en détail les préoccupations des acteurs liées aux risques potentiels de la migration, afin de mieux anticiper et gérer les conséquences du projet.

La seconde est l'initiation des réflexions sur la migration en prenant en compte le processus actuel de développement :

- **Comment extraire la variabilité logicielle à partir des schémas de données des simulateurs ?** Cette question se focalise sur la méthodologie d'extraction de la variabilité logicielle à partir des schémas de données utilisés dans les simulateurs actuels.

La troisième se préoccupe de l'utilisation de la connaissance autour du logiciel pour générer des modèles de caractéristiques :

- **Comment identifier les rôles et les caractéristiques à partir des récits utilisateurs (*user stories*) ?** Cette question examine la manière dont les rôles et les caractéristiques peuvent être identifiés à partir des récits utilisateurs, enrichissant ainsi la compréhension des besoins fonctionnels du logiciel.
- **Comment identifier les contraintes logiques qui pourraient guider la synthèse du modèle de caractéristiques ?** Cette interrogation explore la méthode

permettant d'identifier les contraintes logiques qui influencent la création du modèle de caractéristiques, améliorant ainsi la précision du processus de conception.

Ces questions constituent le fil conducteur du manuscrit et visent à apporter des axes de résolution sur les défis liés à la migration vers une LPL.

1.3 Objectif et contributions

Nous avons observé que notre partenaire industriel ITK applique les bonnes pratiques du développement agile. Cette observation nous a motivés à entreprendre l'étude de la migration de la famille de logiciels existante vers une LPL. Pour cela, nous avons engagé des discussions avec les acteurs du projet afin de comprendre leurs différents points de vue et leurs méthodes de développement. Par la suite, nous avons élaboré une méthode d'ingénierie des LPL qui supporte le processus de migration en partant du système de gestion de versions.

Ces trois axes d'étude ont alors été sélectionnés : l'adoption de la LPL, sa migration, son évolution et sa maintenance.

Adoption de la LPL

Plus un changement est significatif, plus il est susceptible d'être difficile à accepter. Cela s'applique particulièrement aux changements de technologies ou de pratiques. Les utilisateurs peuvent se sentir désorientés ou peu compétents face à la nouveauté. L'étude préalable à l'adoption, dans le cadre de la migration, vise à atténuer cette résistance au changement tout en intégrant les parties prenantes dans l'ensemble du projet. Cette phase comprend des entretiens pour comprendre la perception des avantages et inconvénients d'une LPL. Nous avons pu fournir des lignes directrices et des méthodes pour reproduire des entretiens similaires et mettre en œuvre des solutions concrètes. À partir de ces perceptions nous avons proposé une méthode d'analyse et des lignes directrice pour faciliter l'adoption d'une LPL.

Migration et évolution de la LPL

Pour réaliser une migration cohérente, il est essentiel de comprendre le processus de développement en cours ainsi que les pratiques actuellement en place. Dans le contexte d'ITK, la première phase du développement d'un nouveau produit consiste à intégrer un simulateur en se basant sur ses entrées et ses sorties. Nous avons donc développé une méthode d'analyse de la variabilité de ces simulateurs, ce qui nous permettra de guider la configuration des nouveaux produits et d'identifier les similitudes et les différences entre les produits existants. Au cours de cette étape, nous avons examiné en détail les entrées et les sorties des différents simulateurs. Pour cela, nous avons utilisé plusieurs

méthodes de Traitement Automatique du Langage Naturel (TALN) et d'Analyse Formelle de Concepts (AFC) pour nettoyer les données et identifier la variabilité. Cette analyse constitue la première étape du développement de nouveaux produits au sein de la LPL.

Migration et maintenance de la LPL

Il est nécessaire d'identifier à la fois les caractéristiques communes et spécifiques au sein d'une LPL afin de permettre la création de nouveaux produits. L'ensemble des caractéristiques ainsi que leurs interrelations évolueront tout au long du cycle de vie du logiciel. C'est dans cette optique que nous avons développé une méthode de génération de modèles de caractéristiques à partir de la documentation de la base de code hébergée dans des plates-formes de gestion de versions. Les récits utilisateur jouent un rôle central dans cette méthode, car ils permettent, grâce à leurs liens avec les fusions de code, de localiser de manière implicite les caractéristiques. Pour cette approche, nous avons utilisé des plateformes de contrôle de versions telles que Github ou Gitlab pour obtenir les récits utilisateurs et le code source. Pour traiter ces données, nous avons employé des méthodes de Traitement Automatique du Langage Naturel (TALN) et de classification, ainsi que l'Analyse Relationnelle de Concepts (ARC) afin de construire des modèles de caractéristiques et leurs contraintes.

1.4 Structure de la thèse

Ce manuscrit se consacre à l'étude de différents aspects de la gestion de projet, du développement logiciel et de l'ingénierie des lignes de produits logiciels (LPL). Ces travaux s'intègrent dans l'évolution et la transformation des processus de développement, notamment en ce qui concerne la migration et l'adoption de méthodes agiles. Ce manuscrit est structuré autour de cinq chapitres distincts.

— Le **chapitre 2** offre un aperçu des travaux existants dans les quatre principales thématiques abordées dans l'ensemble du manuscrit. Nous commençons par les études sur l'agilité, en mettant particulièrement l'accent sur son application dans le contexte des LPL. Ensuite, nous abordons la discussion sur l'adoption de nouvelles méthodes et pratiques de développement, telles que les LPL. Puis nous examinons la migration d'architecture et les différentes méthodes d'ingénierie des LPL. Enfin, nous explorons l'utilisation de l'Analyse Formelle de Concepts (AFC) et de l'Analyse Relationnelle de Concepts (ARC) dans l'identification de la variabilité, en mettant en évidence leur pertinence dans le contexte des LPL.

— Le **chapitre 3** présente la première contribution, qui consiste en une étude de la perception de la migration par les acteurs impliqués. Un questionnaire a été élaboré, des interviews ont été menées, les résultats ont été analysés, et des lignes directrices ont été proposées. L'objectif de ces travaux est de fournir un cadre et

des directives pour une migration réussie. Ce cadre et ces directives ont ensuite été appliqués dans le contexte d'une migration vers une LPL, et leur mise en œuvre détaillée est présentée.

— Le **chapitre 4** expose les premiers travaux techniques liés à la migration, réalisée en collaboration avec ITK, notre partenaire industriel. Les nouveaux logiciels développés reposent sur les entrées et sorties des simulateurs élaborés par les agronomes. En suivant cette logique, nous avons examiné la variabilité présente dans les descriptions des entrées et sorties des simulateurs. Pour ce faire, nous avons utilisé des méthodes de Traitement Automatique du Langage Naturel (TALN) pour nettoyer et formater les données, puis l'analyse formelle de concepts (AFC) pour identifier la variabilité. Cette méthode, grâce à l'AOC-Poset, nous a permis de générer un modèle de variabilité représentant les parties communes et spécifiques de chacun des simulateurs.

— Le **chapitre 5** décrit en détail une méthode semi-automatique et extensible de génération de modèles de caractéristiques. Cette méthode propose d'utiliser les récits utilisateur ainsi que leurs liens avec le code source, grâce à une plateforme de gestion de versions et des fusions de code, enrichie par une ontologie pour construire des modèles de caractéristiques. Le processus fait usage du TALN pour décomposer et classifier les récits utilisateurs, et il utilise l'ARC (Analyse Relationnelle de Concepts) pour établir les contraintes des modèles. Ces travaux ont été validés en utilisant des données réelles provenant d'ITK.

— Le **chapitre 6** conclut en résumant les contributions et les résultats et en proposant des perspectives futures.

CHAPITRE 2

État de l'art

Sommaire

Ce chapitre vise à établir les fondements essentiels pour appréhender le contenu de cette thèse. Cet état de l'art a pour double objectif de situer nos travaux par rapport aux recherches existantes et de les ancrer dans un contexte plus global. Nous nous concentrons sur les concepts de l'agilité dans l'ingénierie logicielle, l'adoption de nouvelles pratiques et architectures, ainsi que l'impact de l'Analyse Formelle de Concepts sur le domaine de l'ingénierie logicielle. Chaque section de cet état de l'art est particulièrement axée sur l'ingénierie des LPL.

2.1 Introduction

Les Lignes de Produits Logiciels (LPL) peuvent être perçues comme une famille de produits partageant une base commune tout en présentant des variations. Il existe trois différentes approches dans l'ingénierie des lignes de produits : proactive, extractive et réactive [Kru01]. L'approche proactive implique une ingénierie de domaine visant à modéliser la variabilité, ainsi qu'une ingénierie de l'application pour faciliter la dérivation de variants spécifiques. L'approche extractive vise à extraire rétroactivement les éléments communs à partir de produits existants. Les LPL ont pour objectif d'accroître l'efficacité du développement logiciel en réduisant la redondance et en simplifiant la maintenance, tout en favorisant la création de familles de produits logiciels. L'approche réactive consiste à construire itérativement la ligne de produits par intégrations successives de la variabilité au fur et à mesure que l'on développe de nouveaux produits ou que l'on fasse évoluer les produits existants.

Dans la figure 2.1, une famille de logiciels est représentée, avec chaque logiciel dédié à une culture et composé de différentes caractéristiques correspondant à la présence ou à l'absence de diverses prévisions. Les caractéristiques incluent, par exemple, le type de prévision tel que les maladies ou les récoltes, ainsi que les types de données utilisées pour ces prévisions, comme la météo ou la nature du sol. Sur la figure, le premier niveau comprend les trois logiciels déjà développés pour la vigne, la banane et la tomate. Le deuxième niveau concerne l'architecture de référence, englobant des caractéristiques telles que les prévisions pour les plantes, la météo, le taux de carbone, la nature du sol et les maladies. Le dernier niveau intègre les nouveaux logiciels pouvant être élaborés à partir des combinaisons possibles de ces caractéristiques.

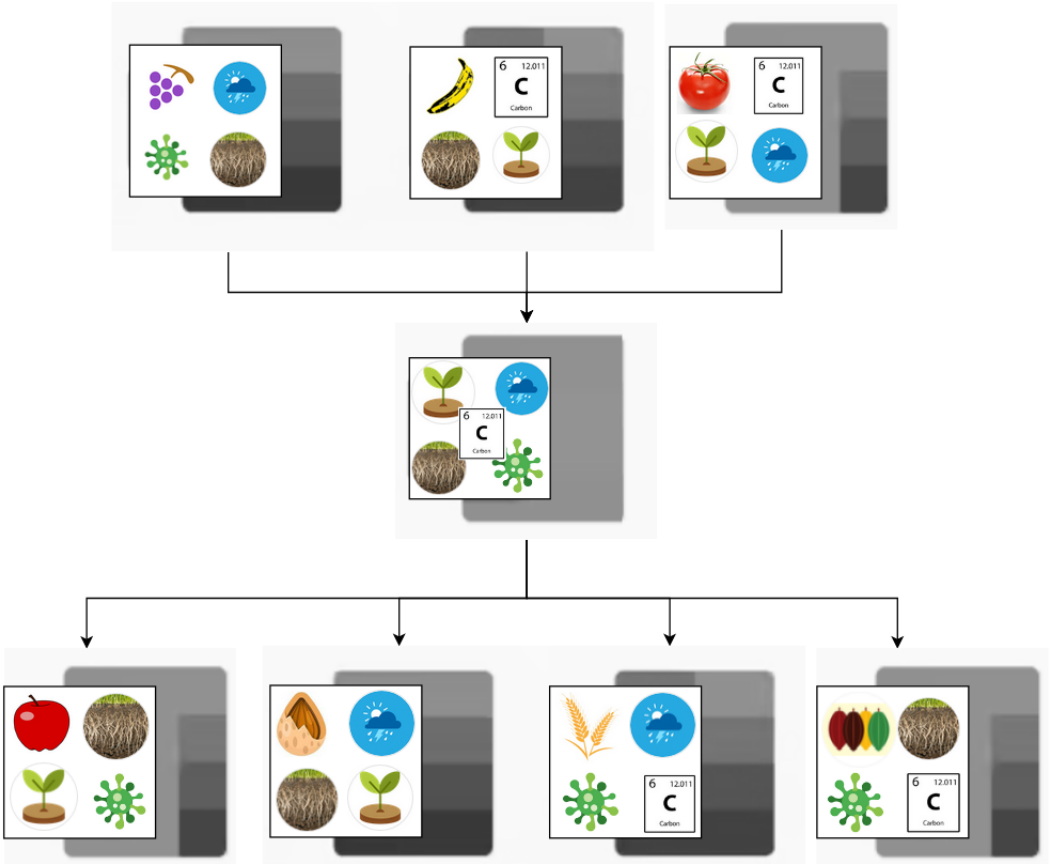


FIGURE 2.1 – Nous envisageons une famille de logiciels, chaque logiciel étant dédié à une culture et composé de différentes caractéristiques correspondant, par exemple, au type de prévision donné : maladies ou récoltes ou aux types de données utilisées pour ces prévisions, comme par exemple, la météo ou la nature du sol

Ingénierie des Lignes de Produits Logiciels L'ingénierie des LPL est un ensemble de méthodes pour la construction de nouveaux logiciels au sein d'un famille de logiciels [vdL02, CN02]. Ces méthodes mettent en place le principe des lignes de produits de manière générale et partagent des objectifs similaires qui visent à optimiser le développement en réduisant les efforts et les délais. En assurant la cohérence et la qualité à travers les produits, une LPL facilite également l'adaptabilité en permettant des personnalisations modulaires ainsi qu'en maximisant la réutilisation des composants communs [PBvdL05]. La gestion centralisée des changements simplifie la maintenance et favorise les économies d'échelle. La LPL peut ainsi libérer des ressources pour se concentrer sur l'évolution ou l'innovation, renforçant la compétitivité et permettant des ajustements rapides en réponse aux évolutions du marché.

Famille de logiciels Une famille de logiciels [Par76,Bos00] est un ensemble de logiciels qui partagent des caractéristiques communes ou similaires en termes de structure et/ou d'architecture. Ces logiciels sont conçus de manière à exploiter des éléments réutilisables et à partager une base de code commune, tout en permettant des variations spécifiques pour répondre aux besoins et aux exigences spécifiques des différents produits ou clients.

Variabilité La variabilité [SJ04, BFG+01] dans les LPL est le caractère d'un logiciel à disposer de variantes ayant des éléments partagés et des éléments distincts parmi lesquels on pourra trouver des exigences, des architectures, des composants et des cas de test. Ce concept est essentiel car il permet la flexibilité nécessaire pour adapter les produits à différents contextes et répondre à des exigences diverses. La gestion de la variabilité [CBA09] englobe les activités qui consistent à représenter explicitement les différentes variations présentes dans les artefacts logiciels tout au long de leur cycle de vie. Cela inclut la gestion des relations et des dépendances entre les différentes formes de variabilité.

Ingénierie du domaine [PBvdL05] Le processus d'ingénierie du domaine vise à définir les aspects communs et la variabilité d'une ligne de produits logiciels, ainsi qu'à créer des artefacts réutilisables pour mettre en œuvre cette variabilité. Il se compose en cinq sous-processus clés. La gestion de produit gère les aspects économiques et stratégiques de la ligne de produits, en définissant sa portée. L'ingénierie des exigences du domaine collecte et documente les exigences communes et variables, tandis que la conception du domaine crée l'architecture de référence pour les applications. La réalisation du domaine se concentre sur la conception détaillée et la mise en œuvre des composants réutilisables, tandis que le test du domaine assure la validation et la vérification des composants, développant également des artefacts de test réutilisables pour faciliter les tests d'application.

Ingénierie d'application [PBvdL05] Le processus d'ingénierie d'application vise à maximiser la réutilisation des éléments de domaine lors du développement de variantes. Il exploite la variabilité et les points communs présents dans la ligne de produits, tout en documentant les exigences, l'architecture, les composants et les tests des applications, et en les reliant aux éléments de domaine. Le processus d'ingénierie d'application fixe le choix des caractéristiques en fonction des besoins de l'application, estime les impacts des divergences entre les exigences de l'application et celles du domaine, et assure la validation et la vérification des applications par rapport à leurs spécifications.

Les liens entre ingénierie du domaine et ingénierie d'application sont illustrés dans la figure 2.2.

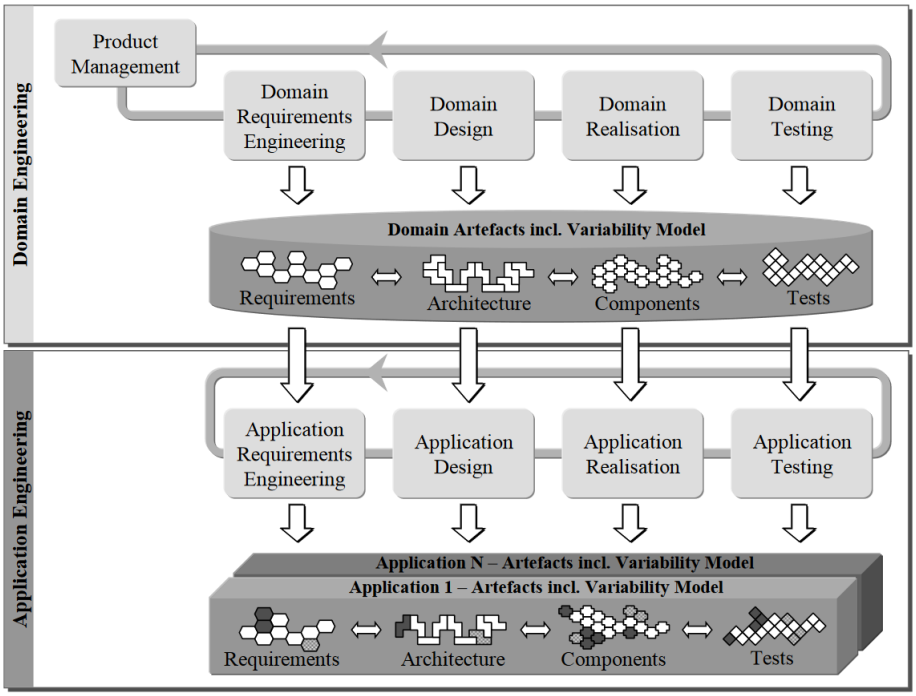


FIGURE 2.2 – Ingénierie du domaine et Ingénierie d'application [PBvdL05]

Caractéristiques Les définitions d'une caractéristique la décrivent comme étant un artefact ou un élément distinctif [KCH+90a] d'un système logiciel. Ces caractéristiques sont définies comme des abstractions fonctionnelles distinctement identifiables qui nécessitent mise en œuvre, tests, livraison et maintenance [KKL+98]. D'un autre point de vue, une caractéristique peut englober tout ce que les utilisateurs ou les programmes clients pourraient souhaiter contrôler concernant un concept [Cza02]. Dans ce contexte, une caractéristique est un ensemble d'exigences fonctionnelles et non fonctionnelles [Bos00]. Du côté utilisateur, une caractéristique peut être vue comme une caractéris-

tique produit constituée d'un ensemble cohérent d'exigences individuelles [CZZM05]. En outre, ces artefacts peuvent introduire de nouveaux services, capacités ou relations [Bat06]. En termes de développement, un artefact peut être une extension de la caractéristique d'un produit [BBC06]. Finalement, une caractéristique peut être comprise comme une structure modifiant le programme initial pour répondre aux exigences des parties prenantes, encapsuler des décisions de conception et offrir des options de configuration [ALB+07]. Ces diverses définitions convergent pour décrire une caractéristique comme un élément distinctif d'un logiciel, qu'il soit visible pour les utilisateurs ou qu'il influe sur les artefacts internes du système. Ce concept nécessite une mise en œuvre soignée ainsi qu'une maintenance continue pour répondre aux besoins des parties prenantes et aux exigences fonctionnelles et non fonctionnelles.

2.2 Agilité et ingénierie des lignes de produits logiciels

L'agilité [BBvB+01] est une pratique de gestion de projets moderne basée sur la proximité avec les clients, l'auto-organisation des équipes et la rapidité des livraisons. Initialement identifiée en Finlande sur des projets réussis dans les bâtiments et travaux publics, l'agilité s'est rapidement propagée à l'informatique avec quelques grandes méthodes comme Scrum [Sch97], eXtreme Programming (XP) [BA05], Kanban [Hir08] ou le Développement Rapide d'Applications (RAD) [Mar91]. Aujourd'hui ces méthodes ont évolué et l'agilité est une pratique répandue dans les projets en informatique. L'agilité a été popularisée pour résoudre les problèmes liés à l'effet tunnel [Mor16] des projets de grande ampleur.

L'ingénierie des LPL et le développement agile sont deux approches bien établies dans le domaine du développement logiciel. Alors que l'ingénierie des LPL vise à maximiser la réutilisation et la variabilité des composants logiciels pour répondre aux besoins multiples des clients, le développement agile met l'accent sur la flexibilité, la collaboration et la satisfaction du client tout au long du processus de développement. Au premier abord, ces deux approches peuvent sembler distinctes, mais l'intégration de l'ingénierie des LPL et du développement agile présente un potentiel significatif pour améliorer l'efficacité, la qualité et la flexibilité du développement logiciel.

2.2.1 Développement agile de logiciels

L'agilité en développement logiciel est une approche fondamentale pour la gestion de projets et la création de logiciels de qualité [Mar03]. Cohen et al. [CLC04] explorent l'histoire des méthodes agiles, notamment le manifeste agile, et discutent de leur signification et de leur rôle dans le développement moderne. Ils offrent également des comparaisons entre différentes méthodes agiles populaires, des guides pour déterminer leur applicabilité, ainsi que des résumés d'études empiriques et d'expériences pratiques.

Méthodologies agiles L'agilité est un ensemble de pratiques qui ont évolué. Il n'y a pas une agilité. La première méthode créée est l'*Extreme Programming* [BA05], ou XP. Ce sont des pratiques du développement logiciel axées sur des objectifs communs et atteignables par toute l'équipe. En utilisant les valeurs et principes de XP, les équipes appliquent les pratiques dans leur contexte. Cela encourage la créativité humaine et vise à produire des logiciels de qualité à un rythme soutenable. XP vise à apporter responsabilité et transparence au développement logiciel, à obtenir de meilleurs résultats avec moins de défauts. Il prend en compte les besoins de toutes les parties prenantes du développement logiciel.

Une méthode plus moderne est *Scrum* [Sch97]. Les méthodes de développement Agile mettent actuellement en avant Scrum comme un leader dans l'adaptation au changement de direction des projets. Scrum propose des pratiques pour produire efficacement et itérativement des logiciels de qualité [SB01]. Les avantages incluent la possibilité de commencer à produire du logiciel de manière incrémentale, quelle que soit la méthodologie existante, ainsi que la simplification de la mise en œuvre des processus Agile, y compris XP. Les pratiques de la méthodologie SCRUM comprennent la mise en place d'un *backlog*, les Sprint plannings, réunions quotidiennes (*daily*) ou les rétrospectives.

Les principales différences entre Scrum et XP résident dans leur approche de la livraison de logiciels fonctionnels. Dans Scrum, les logiciels sont fonctionnels et livrés au cours de courtes itérations appelées « sprints ». La livraison itérative de caractéristiques opérationnelles est privilégiée. XP met davantage l'accent sur l'automatisation des tests et la livraison rapide de logiciels fonctionnels. Avec XP, la qualité des logiciels est privilégiée par le biais de ces tests automatisés.

Crystal Clear [Coc04] est une méthodologie axée sur l'humain pour les petites équipes de développement, mettant en avant la communication efficace et la flexibilité. C'est une méthodologie légère centrée sur les membres de l'équipe pour la construction de logiciels. Elle décrit les rôles, les équipes, les valeurs, les intentions, les habitudes, les activités, les politiques et les produits de travail d'une petite équipe de développement logiciel pour laquelle le temps de mise sur le marché et les coûts de développement sont des considérations essentielles.

Agilité dans la gestion de projets L'approche agile dans la gestion de projet a fait évoluer la manière dont les projets logiciels sont gérés. Contrairement à la gestion de projet traditionnelle, l'agilité permet une plus grande flexibilité et une interaction constante avec les clients tout au long du processus de développement. Cela a conduit à une meilleure adéquation entre les attentes des clients et le produit final. Cependant, l'agilité n'est pas sans ses propres défis, notamment lorsqu'il s'agit de l'adapter à des équipes de grande envergure. La littérature a exploré les avantages de l'approche agile, les conseils pour sa mise en œuvre, ainsi que des solutions pour l'adapter aux grands projets.

L'approche agile permet aux clients d'évaluer la qualité du projet en cours de réali-

sation, offrant ainsi une plus grande transparence et la possibilité d'apporter des modifications en cours de route. Contrairement à la gestion de projet traditionnelle, l'agilité favorise la souplesse et la collaboration, ce qui rend les projets plus réactifs aux changements. Les modifications apportées sont généralement axées sur l'ajout de valeur au projet, ce qui améliore la satisfaction du client.

Larman et al. [Lar02] décrivent la mise en œuvre des méthodes agiles, mettant l'accent sur l'itération et l'amélioration continues. Ils décrivent les pratiques allant de la planification aux tests, ainsi que les valeurs de l'agilité visant à encourager l'amélioration des pratiques de développement. Eckstein et al. [Eck04] abordent quant à eux le défi d'adapter les méthodes agiles aux grandes équipes, en proposant des techniques pour exploiter l'efficacité de l'agilité dans les projets de grande envergure, notamment en abordant les valeurs agiles, la coordination d'équipes multiples et l'impact de la taille du projet sur l'architecture. Les processus agiles ont fait évoluer le développement logiciel en le rendant plus rapide et efficient, permettant d'adapter les projets aux changements de besoins, et en mettant l'accent sur l'aspect humain. Cependant, la plupart de ces méthodes sont conçues pour de petites équipes, ce qui pose problème aux grandes équipes. Ils proposent des techniques pour adapter les méthodes agiles aux équipes de 1 à 200 personnes, voire plus, en abordant des sujets tels que les valeurs agiles, la coordination d'équipes multiples et l'influence de la taille du projet sur l'architecture. Les pratiques agiles sont moins contraignantes sur de petits projets autonomes, mais les responsables font face à des obstacles lorsqu'ils tentent de les intégrer dans des organisations traditionnelles. Ces obstacles sont catégorisés en termes de portée, de processus métier et de conflits entre les personnes [BT05]. Les responsables des projets sont confrontés à des défis nouveaux au sein de grandes organisations cherchant à adopter des approches agiles pour leurs projets.

L'agilité a enrichi la gestion de projet en permettant une plus grande flexibilité, une meilleure interaction avec les clients et une réactivité aux changements. Les conseils et les techniques proposés offrent des solutions pour exploiter les avantages de l'agilité dans des projets de toutes tailles. En fin de compte, l'agilité continue à jouer un rôle dans l'amélioration des pratiques de développement logiciel et dans la gestion de projets réussie [Int20].

Agilité dans la maintenance logicielle Dans le domaine de la maintenance logicielle, l'application des méthodes agiles a suscité un intérêt croissant entre autres grâce à la souplesse qu'elles apportent.

Hanssen et al. [HYCM09] se sont penchés sur les défis spécifiques liés à cette application des méthodes agiles. Ils ont mis en lumière le défi de la « dégradation logicielle », qui correspond au fait que les modifications répétées altèrent progressivement la structure et la qualité du système. Malgré l'accent mis sur l'adaptabilité et la réactivité au changement dans l'agilité, il existe un manque de soutien pour la prise de décision en

matière de refonte et pour l'analyse de la complexité des tâches. Les développeurs ont besoin d'une aide pour comprendre, planifier et tester l'impact des changements. Leur étude propose des stratégies pour gérer la dégradation logicielle dans un contexte agile.

Depuis l'année 2000, les méthodologies agiles ont gagné en popularité, en particulier en matière de maintenance logicielle, considérée comme coûteuse et fastidieuse. Une revue de la littérature de plus de 30 études de recherche menées entre 2001 et 2015 a été réalisée pour évaluer l'efficacité des méthodologies agiles en maintenance [TC16]. Cette revue de la littérature offre des perspectives pour améliorer la qualité et la maintenance des logiciels en agilité. L'application des méthodes agiles à la maintenance logicielle présente des défis, notamment celui de la gestion de la dégradation logicielle. Cependant l'adoption croissante des méthodologies agiles depuis les années 2000 dans ce domaine offre des opportunités d'amélioration de la qualité des logiciels. Les stratégies proposées offrent des pistes pour surmonter ces défis et exploitent l'efficacité de l'agilité dans le contexte de la maintenance logicielle.

Dans l'ensemble, les travaux présentés couvrent un large éventail de sujets liés à l'agilité en développement logiciel, de la gestion de projets à la méthodologie de développement en passant par la maintenance et l'adaptation aux changements. Ils constituent une ressource inestimable pour les professionnels et les équipes cherchant à adopter ou à améliorer leurs pratiques agiles dans le domaine du logiciel.

2.2.2 L'agilité dans les lignes de produits logiciels

L'introduction de l'agilité au sein des LPL existantes est un sujet important dans le domaine du développement logiciel. Cette convergence entre l'agilité et l'ingénierie des LPL offre des opportunités pour améliorer la réactivité aux besoins changeants des utilisateurs tout en tirant parti des avantages de la gestion systématique des produits logiciels. Dans cet état de l'art, nous explorerons plusieurs travaux de recherche qui se penchent sur cette convergence.

Plusieurs travaux de recherche ont exploré cette convergence de manière approfondie. Certains de ces travaux ont examiné comment le développement Agile centré sur le client [BB11] peut être intégré de manière efficace au sein d'une ligne de produits logiciels existante. D'autres ont proposé des modèles spécialement conçus pour l'ingénierie des LPL Agile [GM08], offrant des méthodologies structurées pour gérer les LPL dans un contexte Agile.

Cependant, pour comprendre pleinement cette convergence, il est essentiel de considérer les tendances et les défis identifiés dans ces travaux de recherche. Certains ont réalisé des analyses systématiques des travaux existants, mettant en lumière les opportunités et les défis dans ce domaine en constante évolution [dSdMSNO+11]. D'autres ont effectué des revues systématiques de la littérature, synthétisant ainsi les recherches antérieures et identifiant les tendances émergentes dans le domaine de l'ingénierie des LPL agiles [DPAG11]. Enfin, certains travaux se sont concentrés sur les facteurs clés de

succès pour l'ingénierie des LPL agiles, fournissant des recommandations pratiques pour ceux qui souhaitent mettre en œuvre cette approche innovante, en mettant en évidence les approches, les tendances et les facteurs de réussite [Han11].

Dans le contexte de l'ingénierie des LPL, il est devenu de plus en plus essentiel d'explorer les différentes méthodologies agiles pour les LPL. Cette partie se penche sur plusieurs travaux de recherche qui traitent des LPL et des approches agiles.

Les travaux de McGregor et al. [McG08] visent à décomposer les principes fondamentaux des LPL agiles. Alors que les équipes agiles abordent les changements de manière itérative, les organisations qui réalisent des lignes de produits ont une vision d'investissement axée sur un ensemble de produits. Bien que des contradictions apparentes entre ces deux approches puissent exister, ils remettent en question ces différences et suggèrent la reconstruction d'une méthode hybride qui considère l'agilité comme un attribut de qualité des processus. Dans une perspective similaire, Babar et al. [BIP09] se penchent sur l'architecture des LPL dans le contexte du développement agile. L'intérêt croissant pour l'intégration de ces deux approches complémentaires découle de leurs objectifs partagés. Ce travail présente les résultats d'une étude de cas industrielle montrant comment, malgré des différences philosophiques apparentes, une entreprise de développement logiciel a réussi à fusionner l'architecture des LPL avec les pratiques de développement agile, fournissant des informations utiles sur l'exploitation des pratiques des lignes de produits dans un contexte de développement agile. Noor et al. [NRG08] propose une approche collaborative pour la planification de lignes de produits agiles, illustrée par une étude de cas. L'article se penche sur l'efficacité des méthodes agiles et de l'ingénierie des lignes de produits pour améliorer la satisfaction client et réduire les délais de mise sur le marché. Cette approche demande une réflexion sur la manière dont les principes agiles peuvent soutenir les processus spécifiques des LPL. Les *ThinkLets* sont les pratiques collaboratives qui sont intégrés dans cette approche pour équilibrer l'agilité avec les besoins intrinsèques de la planification de lignes de produits. Une étude de cas avec un partenaire industriel confirme la faisabilité et les avantages de cette approche. Ghanam et al. [GM08] présentent un modèle itératif destiné à l'ingénierie des LPL agiles. L'article explore la convergence potentielle entre le développement agile de logiciels (ASD) et l'ingénierie des LPL (SPLE), bien que ces deux approches semblent initialement distinctes. L'ASD met l'accent sur une forte implication de l'entreprise, tandis que la SPLE privilégie l'analyse détaillée du domaine. L'intégration de l'ASD et de la SPLE offre des opportunités pour améliorer la qualité, réduire les coûts et accélérer la mise sur le marché. Ils proposent un modèle permettant aux organisations agiles d'établir des lignes de produits tout en conservant leur agilité, en utilisant des tests automatisés pour dériver des actifs à partir du code existant.

Ces recherches démontrent l'importance croissante de l'intégration de l'agilité dans l'ingénierie des LPL, soulignant les avantages potentiels de cette convergence et proposant des approches pour surmonter les divergences apparentes entre ces deux domaines.

L'intégration de l'agilité dans le contexte de l'ingénierie des LPL représente un domaine d'intérêt majeur dans le développement logiciel moderne. Cette convergence entre l'agilité et les LPL offre des opportunités significatives pour améliorer la réactivité aux besoins changeants des utilisateurs tout en capitalisant sur les avantages de la gestion systématique des produits logiciels.

Le travail de Mohan et al. [MRS10] se concentre sur les possibilités d'intégrer l'ingénierie des LPL avec les méthodologies agiles dans le développement logiciel. L'article met en évidence les avantages économiques de la gestion de LPL pour les organisations de développement logiciel et explique comment les méthodologies agiles mettent l'accent sur l'improvisation, la réactivité aux changements et la considération des individus en tant qu'élément central dans le processus de développement. Dans leur article, Hanssen et al. [Han11] explorent les facteurs qui facilitent l'intégration de l'ingénierie des LPL dans un environnement agile. Ils remettent en question l'idée préconçue selon laquelle ces deux approches, bien que prometteuses, seraient en conflit en raison de leurs objectifs différents. Au contraire, ils démontrent que ces approches peuvent se compléter mutuellement. En proposant un cadre conceptuel, ils offrent des pistes pour réaliser le travail stratégique, tactique et opérationnel nécessaire au développement de lignes de produits dans un contexte agile. L'intégration de l'agilité aux LPL au sein de grandes entreprises est un challenge [KHPS19]. Cette intégration a été étudiée ainsi que la transition vers l'ingénierie des LPL agiles dans de grandes entreprises. De nombreuses grandes entreprises sont confrontées à la complexité croissante des systèmes logiciels, ce qui les pousse à adopter des LPL pour gérer efficacement la réutilisation et la complexité. Cependant, ces lignes de produits semblent parfois manquer de réactivité aux changements. Les approches de développement agile promettent de répondre à cette problématique, mais leur intégration peut être complexe.

L'intégration de l'agilité et de l'ingénierie des LPL est un domaine de recherche en constante évolution, offrant des perspectives intéressantes pour le développement logiciel moderne. Les travaux de Mohan et al. et Hanssen et al. mettent en évidence les avantages potentiels de cette convergence et ouvrent la voie à de nouvelles approches pour optimiser le processus de développement de produits logiciels dans un environnement en constante évolution.

2.2.3 Agilité à ITK

Chez notre partenaire industriel, les équipes mettent en œuvre la méthode agile Scrum pour optimiser leur travail. Cette approche repose sur un cycle de trois mois appelé le "*Program Increment (PI) Planning*". Pendant cette période, les équipes organisent trois à quatre "*sprints*", chacun avec sa propre planification détaillée. À la fin de chaque sprint, une rétrospective est effectuée pour évaluer ce qui s'est bien passé et ce qui pourrait être amélioré. Chaque jour les membres des équipes se réunissent pour un "*Daily Meeting*" pour discuter des difficultés et points de blocage.

L'agilité est aussi appliquée entre les équipes avec des "*mercatos*" organisés afin d'offrir aux membres des équipes l'opportunité de changer d'équipe. Ceci produit un effet dynamisant au sein des salariés de l'entreprise, leur permettant de changer d'interlocuteurs au cours des projets. Cette organisation agile s'inscrit dans le cadre du framework SAFe [Lef07]. SAFe est un ensemble de modèles organisationnels et de méthodes de travail visant à mettre en œuvre des pratiques agiles à l'échelle de l'entreprise. Nous avons porté notre attention sur les pratiques de développement au sein de ce framework, en nous focalisant plus particulièrement sur celles employées au sein de l'entreprise ITK. Pour mettre en œuvre une nouvelle caractéristique, un récit utilisateur est rédigé dans une plateforme de gestion de tickets telle que *Jira* ou *Trello*, ou dans une plateforme de gestion de versions telle que *Gitlab*, *Github*, ou *Bitbucket*. Ce récit utilisateur décrit les besoins du point de vue de l'utilisateur. Ensuite, les développeurs décomposent ce récit utilisateur en tâches plus techniques et procèdent à l'implémentation de la caractéristique en utilisant le logiciel de gestion de versions *Git* et en effectuant des fusions de code (Merge). Cette approche permet de garantir une meilleure traçabilité des besoins des utilisateurs tout en facilitant la collaboration entre les membres des équipes de développement.

2.3 Adoption des lignes de produits logiciels

Les avancées technologiques sont souvent perçues initialement comme incertaines et risquées [Rog83]. Pour surmonter cette incertitude, la plupart des individus cherchent des projets similaires aux leurs et qui ont déjà adopté la nouvelle idée. Cela établit un processus de diffusion, qui débute avec quelques individus pionniers qui adoptent en premier une innovation, puis la propagent, un processus qui peut s'étaler sur des mois voire des années.

L'adoption de projets d'avancées technologiques revêt une importance cruciale dans l'évolution du domaine de l'ingénierie logicielle [CS09]. L'étude de l'adoption met en lumière la maturité du domaine de l'architecture logicielle, qui englobe maintenant une multitude de notations, d'outils et de techniques d'analyse. Elle est devenue un élément essentiel de la conception et du développement de systèmes logiciels, marquant ainsi l'âge d'or de l'architecture logicielle. Depuis les années 1980, l'architecture logicielle a évolué, passant des descriptions qualitatives empiriques des systèmes utiles à une discipline complète. Initialement axée sur l'interprétation des pratiques logicielles, elle fournit aujourd'hui des directives concrètes pour la conception de logiciels complexes, passant ainsi de la recherche fondamentale à un pilier de la conception logicielle. De manière plus spécifique, il y a de multiples facteurs qui influent sur l'adoption [MB14]. MacLennan et al. combinent la théorie de la diffusion des innovations avec le cadre technologie-organisation-environnement pour élaborer un modèle d'adoption de l'architecture orientée services (SOA). Une enquête menée auprès des architectes d'entreprise en Afrique du Sud a révélé les risques, les obstacles et les avantages liés à l'adoption de la SOA. Ces facteurs incluent l'utilisation de normes multiples, la compatibilité, le soutien de la direction, la gouvernance, les ressources humaines et financières, ainsi que le soutien des fournisseurs, et sont liés à la perception du succès de la mise en œuvre de la SOA par les départements informatiques. Cette approche peut s'étendre pour des réflexions à d'autres types d'architecture. La migration de logiciels est une tâche fondamentale et complexe, de plus en plus pertinente en raison de la quantité des technologies Web et mobiles. Dans le cadre d'un projet de recherche italien, l'article [TPR+11] présente les résultats d'une enquête menée auprès de 59 entreprises italiennes du secteur des technologies de l'information, se penchant sur leurs expériences en matière de migration de logiciels, leurs principaux objectifs de migration et les technologies adoptées. Cette enquête s'est concentrée sur les projets de migration internes vers le Web, les architectures orientées services et les environnements sans fil. Ils concluent que l'automatisation et les processus rigoureux sont limités et qu'il est nécessaire d'encourager la coopération entre l'industrie et le milieu académique pour améliorer les outils de migration. L'industrie du logiciel est de plus en plus confrontée à l'évolution de l'architecture au sein des plate-formes d'écosystèmes industriels [PSZ18], caractérisées par une multitude de solutions tierces et une complexité élevée en termes de conception et de choix technologiques. Les architectes logiciels travaillant sur la migration de plate-formes et

les scénarios d'évolution ont besoin de support pour sélectionner et utiliser les offres optimales, garantir la compatibilité avec diverses contraintes techniques et non techniques, et optimiser les architectures. L'adoption de ces outils de soutien joue un rôle essentiel dans le succès des projets de migration et d'évolution des plate-formes.

L'adoption d'avancées technologiques dans le domaine de l'ingénierie logicielle est une problématique importante, et il est nécessaire de prendre en compte les facteurs déterminants qui influencent cette adoption dans des contextes variés.

2.3.1 Méthodologies pour l'adoption des LPL

L'ingénierie des lignes de produits logiciels est un domaine d'étude important pour améliorer la réutilisation, l'efficacité et la qualité du développement logiciel. L'adoption réussie d'une LPL nécessite une méthodologie robuste et des stratégies appropriées.

L'article de Bayer et al. [BFK+99] présente PuLSE, une méthodologie de développement de LPL. PuLSE se distingue par sa focalisation centrée sur les produits, sa personnalisation des composants, son échelle de maturité pour l'évolution structurée, et ses adaptations à différentes situations de développement. Cette méthodologie vise à simplifier la conception et le déploiement de lignes de produits logiciels dans divers contextes d'entreprise. Bockle et al. [BMK+02] se concentrent sur l'adoption et l'institutionnalisation d'une culture de LPL. Ils fournissent des stratégies pour effectuer la transition vers l'ingénierie des lignes de produits et expliquent comment élaborer un plan d'adoption et institutionnaliser cette approche au sein d'une organisation. Les différentes stratégies présentées sont notamment l'élaboration de scénarios commerciaux détaillés pour évaluer les activités, les efforts et les coûts du point de vue des principales parties prenantes, la création d'un plan d'adoption intégrant les activités nécessaires à la transition et institutionnalisation de l'ingénierie de lignes de produits dans l'organisation. Mansell [Man06] partage des expériences sur l'introduction de la réutilisation systématique dans les petites et moyennes entreprises (PME). Bien que la réutilisation systématique promette des avantages commerciaux importants, les PME sont souvent confrontées à des obstacles pour l'adopter. Mansell aborde les défis et les attentes liés à la réutilisation systématique dans le développement logiciel, en se concentrant sur les domaines organisationnels favorables à cette approche, les bénéfices obtenus, les risques identifiés, les attitudes envers le risque lié à l'investissement, et l'état de l'infrastructure et du soutien au sein des organisations. Simon et al. [SE02] se penchent sur l'introduction évolutive des LPL. Ils explorent la transformation de produits existants en lignes de produits logiciels, une question cruciale pour de nombreuses entreprises. Pour ce faire, ils commencent par analyser le code hérité afin d'évaluer quelles parties du logiciel peuvent être restructurées pour répondre aux besoins de la ligne de produits à des coûts raisonnables. Cette analyse inclut l'utilisation de l'analyse des caractéristiques. Northrop [Nor04] propose une roadmap pour l'adoption des LPL. L'article met en évidence les avantages de l'approche LPL et propose une transition progressive comme une option attrayante pour

réduire les risques et les coûts. Kuvaja et al. [KSH08] fournissent des directives basées sur une étude de cas. Ils démontrent qu'il est possible d'adopter une ligne de produits logiciels de manière efficace avec des investissements initiaux limités. Les directives incluent la stratégie "*starting from scratch*", construisant tout depuis le début avec des investissements initiaux plus élevés mais un temps de développement plus court, ou la stratégie "*exploiting existing systems*", tirant parti des systèmes existants pour réduire les coûts cumulatifs plus rapidement.

L'adoption de lignes de produits logiciels est un domaine en constante évolution, avec diverses méthodologies et approches disponibles. Chaque méthode discutée dans ces articles offre des perspectives sur la manière de faciliter cette adoption. Les organisations doivent choisir la méthode qui convient le mieux à leur contexte et à leurs besoins. Les recherches continues dans ce domaine visent à améliorer encore l'efficacité et la réussite de l'adoption des LPL.

2.3.2 Expériences lors de l'adoption de LPL

L'adoption de l'approche d'ingénierie de LPL dans des contextes industriels et académiques présente un ensemble complexe de défis et d'opportunités.

Une première expérience est rapportée dans le contexte de l'introduction d'une approche de ligne de produits dans le système de commutation vocale S12 d'Alcatel [ES03]. Cet exemple met en lumière les défis pratiques rencontrés lors de cette transition, ainsi que les astuces et les écueils à éviter. Les principes clés de l'ingénierie logicielle, associés à des preuves empiriques et des techniques pratiques, sont présentés. De plus, les difficultés liées à l'introduction de concepts LPL dans des produits existants sont abordées, soulignant que de nombreuses industries sont freinées par ce processus complexe. Une autre étude se penche sur l'adoption des LPL par Bosch Gasoline Systems [STB+04]. Dans l'industrie automobile, l'adoption d'une approche de ligne de produits a permis d'améliorer les processus et produits, avec des stratégies spécifiques pour les logiciels de contrôle moteur. Face à des clients exigeants, la négociation des exigences logicielles pour chaque produit est incontournable. La personnalisation client limite la vente de solutions prédéfinies, mais répond à la demande croissante de logiciels spécifiques et de partage au niveau du code objet. L'innovation rapide est cruciale dans ce marché concurrentiel, générant un flux continu de demandes de modification. La gestion des systèmes de contrôle moteur, devant couvrir de nombreuses variantes, implique la gestion de centaines de versions de programmes annuellement. Le groupe Siemens a également partagé ses expériences, présentant les défis et les meilleures pratiques lors de la transition vers une approche de famille de produits logiciels [KSG06]. Leur proposition se matérialise sous la forme d'un manuel complet offrant des conseils sur la manière de débuter une telle démarche. En parallèle, dans le contexte des compétences requises, il est constaté que la compréhension de la théorie derrière les familles de produits logiciels peut s'avérer complexe et encore plus difficile à appliquer. Bien que l'implémentation de la varia-

bilité soit un aspect maîtrisé, l'étude conclut à la nécessité de se concentrer de manière intensive sur d'autres domaines de recherche. L'expérience acquise sur les étapes d'introduction, l'approche de la modélisation de domaine et de caractéristiques, la discussion sur l'identification et l'analyse des exigences via le traitement du langage naturel, ainsi que les suggestions concernant les rôles du développement, de la gestion de produits et de l'architecture sont présentées comme des domaines à explorer davantage. Une perspective essentielle à considérer est l'adoption des LPL par de petites organisations. Des études montrent que même les petites entreprises ont un potentiel à exploiter en matière d'adoption des pratiques SPL [NR16]. Les avantages potentiels en termes de rapidité de développement et d'efficacité incitent à explorer cette voie. Une réticence à adopter l'approche LPL persiste dans certaines entreprises en raison de l'absence d'un chemin d'adoption clair et de la nécessité de transformations organisationnelles majeures. Pour illustrer que l'adoption réussie est possible, Danfoss Drives partage ses dix années d'expérience en ingénierie de lignes de produits [FSK+16]. Leurs conclusions soulignent le succès de l'ingénierie LPL chez Danfoss, mettant en avant une adoption réactive et progressive, l'enrichissement des pratiques d'ingénierie par l'ingénierie de domaine, une combinaison judicieuse avec le développement agile, et une amélioration de la qualité. Les facteurs clés de réussite incluent une équipe de support dédiée, des réunions de planification régulières, des ateliers et des formations, ainsi qu'une introduction contrôlée des caractéristiques. L'adoption des LPL dans de petites organisations est examinée par Bastos et al. [BdMSNO+17]. Cette étude met en évidence le besoin de directives pour soutenir l'adoption de LPL dans de telles structures. Elle combine une cartographie, une étude de cas et une enquête auprès d'experts pour examiner cette adoption. Un autre exemple intéressant concerne l'adoption des LPL dans l'industrie de l'assurance, où l'ingénierie des LPL est encore peu répandue [Brö18]. L'article présente une stratégie de transition basée sur un projet pilote impliquant plusieurs compagnies d'assurance en tant que clients testeurs, visant à introduire l'ingénierie de lignes de produits. En ce qui concerne l'évaluation de la préparation d'une organisation à l'adoption des LPL, Tuzun et al. proposent un modèle d'aide à la décision [TTKB15]. Ils mettent en avant l'importance d'une assistance automatisée pour soutenir le processus de prise de décision dans ce contexte. Niemela et al. explorent les différentes stratégies d'adoption, mettant en lumière six façons de définir les objectifs et huit stratégies pour les atteindre [Nie05]. Cette diversité d'approches souligne la complexité du choix d'une stratégie appropriée pour une entreprise donnée. Enfin, Fritsch et al. se penchent sur l'analyse du potentiel pour le développement de l'architecture de famille de produits [FH04]. Leur approche consiste en une analyse du potentiel de LPL permettant de décider de la pertinence de l'approche des ligne de produits pour un ensemble donné de produits et un marché cible.

En somme, ces expériences et études offrent une vision approfondie des défis, des astuces et des stratégies liés à l'adoption de l'approche LPL. L'enrichissement des pratiques d'ingénierie, la gestion efficace des exigences, la réactivité aux demandes du marché et la

combinaison judicieuse avec d'autres méthodologies, telles que le développement agile, émergent comme des éléments clés pour le succès de cette transition. Partager ces expériences est essentiel pour guider d'autres organisations dans leur parcours vers les lignes de produits logiciels. Dans ce manuscrit, nous partageons également notre expérience d'adoption d'une LPL au sein de notre partenaire, l'entreprise ITK. Nos travaux sur la perception d'un changement de pratiques s'inscrivent dans ces retours d'expérience.

2.3.3 Frameworks d'évaluation des initiatives LPL

L'évaluation des initiatives en cours dans le domaine de l'ingénierie des LPL est important pour mesurer les avantages potentiels et les risques associés. Il existe différents cadres répondant à ces questions.

Schmid et al. [SJ02] explorent le développement d'une méthode d'évaluation des avantages et des risques des LPL. Cette analyse du potentiel des LPL vise à déterminer rapidement si cette approche est adaptée à un ensemble de produits et à un marché donné. Ils soulignent le manque actuel d'assistance dans ce domaine et présentent un cadre d'évaluation basé sur un atelier d'une demi-journée. Cet outil structuré repose sur un questionnaire, évaluant les produits, les logiciels, les marchés et les clients, et permet de prendre des décisions claires sur la pertinence de l'adoption de LPL. Linden et al. [vdLBK+04] se penchent sur l'évaluation des familles de produits logiciels, un élément clé des initiatives LPL. Ils décrivent un cadre d'évaluation en quatre dimensions liées aux préoccupations de l'ingénierie logicielle, couvrant les aspects commerciaux, architecturaux, organisationnels et de processus. Ce cadre est conçu pour être utilisé au sein des organisations de développement logiciel afin d'évaluer leur propre ingénierie de familles de produits logiciels, de définir des priorités d'amélioration et d'établir des feuilles de route. Pour évaluer la maturité commerciale des LPL, Ahmed et al. [AC15a, AC15b] proposent un modèle de maturité. Cet article met en évidence l'importance de l'alignement entre les aspects commerciaux et l'ingénierie de LPL, tout en fournissant une méthodologie d'évaluation pour mesurer la maturité de la dimension commerciale au sein d'une organisation. Le modèle examine la coordination entre les pratiques commerciales et l'ingénierie de LPL, contribuant ainsi à l'alignement des aspects commerciaux avec les initiatives LPL. Ils présentent un cadre d'évaluation des processus pour les LPL. Cet outil d'évaluation, baptisé SPLPAT, repose sur la logique floue et vise à guider les organisations dans le développement et la gestion réussis des LPL. Il offre une méthode directe pour évaluer le niveau de maturité actuel du processus, tout en traitant l'imprécision et l'incertitude présentes dans les variables du processus logiciel. Dans un contexte similaire, Ahmed et al. [AC10] se concentrent sur un modèle de maturité organisationnelle pour l'ingénierie des LPL. Ils mettent en avant l'interdisciplinarité de cette approche, couvrant les dimensions commerciales, architecturales, de processus et organisationnelles. Le modèle de maturité évalue la dimension organisationnelle et suppose que les aspects comportementaux et de gestion jouent un rôle essentiel dans l'institutionnalisa-

tion de l'ingénierie des LPL au sein d'une organisation. Les questionnaires d'évaluation et la méthodologie de notation sont utilisés pour évaluer la maturité de la dimension organisationnelle. Ces frameworks d'évaluation des initiatives LPL offrent une variété d'approches pour mesurer la pertinence, la maturité et les avantages potentiels de l'ingénierie des LPL. Ils contribuent à l'amélioration de ces initiatives en fournissant des méthodologies structurées pour l'évaluation, l'alignement et le développement réussi des LPL au sein des organisations. Nos travaux font suite à l'évaluation de la pertinence d'une migration. Les décisions d'effectuer une migration ont été prises en amont de nos travaux.

2.3.4 Obstacles à une adoption réussie

L'adoption de l'ingénierie des LPL est une démarche prometteuse pour réduire les coûts, améliorer la qualité et accélérer le développement de logiciels. Cependant, malgré les avantages mesurables, plusieurs obstacles entravent son adoption. Ce texte examine les principaux obstacles à une adoption réussie des LPL, avec des contributions de divers auteurs.

Catal et al. [Cat09] se penchent sur les obstacles à l'adoption de l'ingénierie des LPL. Malgré les avantages des LPL, son adoption est plus lente que d'autres tendances technologiques. Ils examinent les obstacles sous trois points de vue : le commanditaire du projet, l'organisation et les utilisateurs de la LPL, offrant des suggestions pour résoudre ces problèmes multidimensionnels à court terme. Les principales barrières à l'adoption de l'ingénierie de LPL comprennent la focalisation sur la technologie, les différences de terminologie, le manque de ressources pratiques, les problèmes d'intégration matérielle/logicielle, les incertitudes, la nécessité d'un changement organisationnel, le manque de connaissances des sponsors de projet sur la SPLE et le manque d'experts SPLE avec des coûts élevés de formation. Jha et al. [JO09] se concentrent sur les problèmes liés à la réutilisation de logiciels dans les LPL. L'un des principaux objectifs de l'introduction des LPL est la réduction des coûts par la réutilisation d'actifs logiciels. Cependant, la réutilisation de logiciels présente des défis en raison de la complexité des interactions et du nombre croissant de variantes de produits. Ils présentent ces obstacles ainsi que les résultats sur la réutilisation de logiciels dans les LPL, mettant en lumière les problématiques et les préoccupations liées à cette pratique.

Bastos et al. [BdMSNdAdLM11,BdMSNO+17] proposent une étude sur l'adoption des LPL, identifiant les barrières et les défis rencontrés. Ils se concentrent spécifiquement sur l'adoption des LPL dans de petites organisations. Ils examinent les preuves disponibles sur l'adoption des LPL, identifiant les écarts entre les stratégies requises, les structures organisationnelles, le niveau de maturité et les obstacles à l'adoption existants. Cette recherche vise à établir des lignes directrices pour l'adoption des LPL dans des petites et moyennes organisations. Les résultats clés de leurs études sur l'adoption de LPL dans les PME mettent en avant l'importance de la définition d'une stratégie de transition combi-

née, en soulignant l'influence de la taille du domaine, de la maturité organisationnelle et de la structure organisationnelle sur le processus d'adoption. La maturité organisationnelle, joue un rôle déterminant, tandis que les obstacles à l'adoption, plus fréquents dans le contexte des PME, sont directement liés à la stratégie de transition et à la structure organisationnelle. En conclusion, un plan d'adoption bien défini et un partenariat entre les PME et les centres de recherche sont identifiés comme des facteurs essentiels pour le succès du processus d'adoption de LPL.

En conclusion, ces articles offrent un aperçu des obstacles couramment rencontrés lors de l'adoption de l'approche LPL. Ils mettent en lumière les obstacles liés à la réutilisation de logiciels, à la complexité des processus et à l'alignement organisationnel. Ces informations peuvent servir de référence pour surmonter ces défis et à réussir dans l'adoption des LPL. Lors de nos travaux avec notre partenaire ITK, il a été important de prendre en compte les différents aspects de l'adoption et de partager notre expérience dans ce domaine.

2.4 Migration vers les lignes de produits logiciels

La migration de logiciels est une opération complexe visant à déplacer des algorithmes, des services, des données, voire des systèmes d'exploitation d'un environnement à un autre. Elle joue un rôle essentiel dans la maintenance, l'évolution et la flexibilité des systèmes informatiques.

Les défis de la migration logicielle ont conduit à l'exploration de diverses approches pour rendre ce processus plus efficace. Une comparaison avec les méthodes de migration a révélé l'efficacité d'un schéma de migration verticale d'algorithmes et de fonctions, basé sur la modularité du code [PI93]. Cette approche a démontré sa capacité à accélérer la migration tout en minimisant l'utilisation de la mémoire. Dans le contexte de la migration de services cloud, des efforts ont été déployés pour réduire l'effort nécessaire pour migrer entre différents fournisseurs de cloud, en particulier en raison de l'hétérogénéité des API et des bibliothèques. Pour pallier cela une approche repose sur l'adaptation logicielle [MGMC13] pour résoudre le problème du verrouillage du fournisseur, permettant ainsi aux développeurs de créer des applications cloud indépendantes de la plateforme. La gestion de données lors de la migration est un autre domaine clé. Des stratégies de migration de données auto-adaptatives ont été développées pour soutenir le développement logiciel agile, notamment dans le contexte des bases de données NoSQL sujettes à des évolutions fréquentes de modèle de données [HSNK22]. Ces stratégies s'ajustent automatiquement aux besoins de l'application, minimisant ainsi la surcharge liée à la gestion de données versionnées. La qualité de la migration logicielle est également un aspect crucial. Un modèle de qualité a été développé pour évaluer la qualité des systèmes migrés en fonction des outils utilisés [PJW14]. Cette approche est utilisée pour prendre des décisions éclairées concernant les stratégies de migration. Une enquête me-

née en Italie a souligné l'importance de la migration de logiciels, en particulier vers le Web et les architectures distribuées, en raison de l'utilisation croissante d'Internet et des appareils mobiles [TPR+08, TPR+11]. Des approches innovantes, telles que l'automigration en temps réel des systèmes d'exploitation, ont été développées pour améliorer l'utilisation des ressources et minimiser les interruptions lors de la migration [HJ04]. L'automatisation joue un rôle clé dans ces approches, car la migration manuelle peut être coûteuse et inefficace. Enfin, l'ingénierie logicielle orientée modèle a été explorée comme un moyen de faciliter la migration des applications de gestion de l'énergie d'une plateforme à une autre, en transformant automatiquement les modèles de conception en programmes [NWM19].

Ces approches mettent en lumière les défis, les avantages et les opportunités répondant aux besoins changeants de l'industrie informatique. Les travaux présentés dans cette thèse couvrent plusieurs aspects de la migration des LPL et se placent au sein des travaux existants présentés dans cette partie.

2.4.1 Clonage dans les familles de logiciels

Le clonage de code est une pratique courante dans le développement logiciel, permettant de réutiliser des fragments de code existants. Bien que de nombreuses techniques aient été développées pour détecter, comprendre et éliminer les clones de code, d'importants défis subsistent, notamment dans le contexte des LPL. Le clonage et les LPL sont deux approches qui peuvent coexister [EPPC21], le clonage se base sur la génération de familles de produits logiciels à partir de produits existants avec de légères modifications. D'autre part, les LPL reposent sur la réutilisation systématique d'un ensemble de composants logiciels pour dériver de nouveaux produits logiciels

La réutilisation de logiciels réduit les coûts de développement et améliore la qualité des systèmes logiciels [KB21]. Deux stratégies sont courantes : le clonage et l'appropriation (copie et adaptation d'un système) et la réutilisation orientée plateforme (construction d'une plateforme configurable). La première est immédiatement disponible, flexible et initialement peu coûteuse, mais elle ne permet pas de s'adapter à la fréquence de la réutilisation, ce qui entraîne des coûts de maintenance élevés. La seconde permet de s'adapter à la fréquence de la réutilisation, mais elle impose des investissements initiaux importants pour la construction de la plateforme et réduit la flexibilité. En tant que telle, chaque stratégie présente des avantages et des inconvénients distincts, imposant des activités de développement et des architectures logicielles différentes. Le choix d'une stratégie est une décision clé ayant un impact à long terme sur le développement logiciel d'une organisation. Le développement axé sur la réutilisation orientée plateforme est plus coûteux, mais il réduit simultanément les coûts de réutilisation, et il permet d'obtenir une meilleure qualité du code par rapport au clonage et à l'appropriation [KB21]. Le clonage de code ne peut pas être simplement réduit à une solution technique. Il doit être interprété et compris dans un contexte plus large, tenant compte des raisons pour

lesquelles les pratiques de clonage sont adoptées. Malheureusement, il existe encore un manque d'approfondissement de cette compréhension, en particulier du point de vue personnel et organisationnel. Cela limite la capacité à fournir un soutien efficace pour atténuer les problèmes de maintenance causés par les clones de code [ZPXZ12]. Le clonage de code dans le développement de logiciels est influencé par des facteurs ajustables et des points critiques qui affectent l'introduction, l'existence et la suppression des clones. Ces facteurs et points critiques révèlent des opportunités pour améliorer les pratiques de clonage dans le développement industriel du point de vue technique, personnel et organisationnel. De nombreuses entreprises développent des LPL en clonant et en adaptant les artefacts des variantes de produits existants [DRB+13]. Le clonage est souvent perçu comme une approche de réutilisation naturelle par les développeurs, en raison de sa simplicité, de sa disponibilité et de l'indépendance qu'il offre. Cependant, ces avantages sont accompagnés de problèmes qui limitent l'adoption d'approches de réutilisation systématique des logiciels. Il est essentiel de comprendre comment le clonage est utilisé dans le contexte des LPL et comment il peut être optimisé pour améliorer la qualité des produits logiciels.

Les pratiques industrielles courantes conduisent souvent au développement de produits logiciels similaires, gérés de manière ad hoc, ce qui entraîne progressivement une faible qualité du produit [ZHP+14]. Dans de tels cas, la migration vers une LPL peut s'avérer nécessaire pour surmonter ce problème. En résumé, le clonage de code est un aspect du développement de LPL, et son impact doit être étudié, en tenant compte des aspects techniques, personnels et organisationnels pour optimiser la réutilisation de code et améliorer la qualité des produits logiciels. Chez notre partenaire industriel, l'entreprise ITK, les pratiques de développement actuelles consistent principalement en un clonage ad hoc de nouveaux produits à partir des simulateurs. Nos travaux visent à proposer des alternatives en intégrant ces pratiques actuelles en vue d'une migration vers les LPL.

2.4.2 Migration des LPL

La migration d'un système logiciel vers une architecture basée sur des LPL peut être un processus complexe. Différentes approches peuvent être utilisées pour faciliter cette transition. Dans cette section, nous explorerons trois approches principales : proactive, extractive et réactive. La définition de la portée d'une LPL est nécessaire pour délimiter les objectifs, les produits et les domaines, en se concentrant sur les aspects techniques et organisationnels de la réutilisation [MRA+22].

Méthodologies et outils

La gestion des LPL est un défi complexe dans le domaine du développement logiciel. Pour répondre aux besoins de personnalisation, de réutilisation et de gestion des configurations complexes d'applications, diverses approches, méthodologies et outils ont été

développés.

La méthodologie PuLSETM [BFK+99] est une approche centrée sur le produit pour la conception et le déploiement de LPL. Contrairement aux approches classiques d'ingénierie de domaine, PuLSETM favorise la personnalisation des composants, l'introduction progressive, et l'évolution structurée. Cette méthodologie est le fruit des leçons tirées de collaborations industrielles. WipeOutR [LFU+22] est une approche visant à améliorer la qualité des modèles de caractéristiques utilisés dans la spécification des propriétés de variabilité et de l'ensemble des artefacts logiciels. Elle identifie les redondances au sein des modèles de caractéristiques, améliorant ainsi les performances de configuration. L'approche fMultiLTL [DLCL23] permet d'exprimer des propriétés multiples au sein des familles de systèmes en prenant en compte plusieurs traces issues de différentes variantes d'une SPL. Cette approche renforce la qualité et la robustesse des LPL. L'approche SMartyCheck [BHM+23] vise à améliorer la qualité des artefacts logiciels au sein des LPL en identifiant et corrigeant les défauts. Étant donné que les artefacts sont réutilisés pour plusieurs produits spécifiques au sein d'un LPL, cette approche a un impact significatif. L'utilisation des bascules de caractéristiques [BWZ15] (*feature flipping* ou *feature toggles* ou *feature flags*), pour gérer la variabilité des caractéristiques dans le code source peut être un premier pas vers une migration [JKA22a]. L'objectif est d'unifier les concepts de résolution de la variabilité pendant la conception et au moment de l'exécution en utilisant un modèle de caractéristiques [NKSB19]. Ce modèle permet une résolution partielle au moment de la conception, générant ainsi des bascules de caractéristiques activables ou désactivables en temps réel. Une possibilité pour automatiser les adaptations nécessaires dans la base de données pour modifier la sélection de caractéristiques d'un produit existant serait un modèle d'évolution [CLPP23]. Ce modèle assure la traçabilité entre les caractéristiques et les éléments du modèle de données affectés. FeatureIDE [MTS+17] est un environnement de développement open-source conçu pour la modélisation et la mise en œuvre des LPL. Il permet l'analyse des systèmes en utilisant des modèles de caractéristiques, la spécification des exigences sous forme de configurations, et la génération de code basée sur la compilation conditionnelle et la programmation orientée caractéristiques. BUT4Reuse [MZB+15] répond au défi que rencontrent les développeurs pour gérer des ensembles existants de variantes d'artefacts logiciels. Il propose un cadre qui intègre divers algorithmes pour identifier les caractéristiques, localiser les caractéristiques, découvrir les contraintes liées aux caractéristiques, et mettre en œuvre des approches de réingénierie. Les principes de ce cadre sont conçus pour être génériques et extensibles, avec la possibilité d'ajouter des adaptateurs, des algorithmes et des visualisations pour s'adapter à différents scénarios. The ECCO Tool [FLLE15] répond à un besoin en matière de réutilisation de logiciels, particulièrement dans le contexte de la pratique courante de clone-and-own, où les entreprises copient des systèmes existants pour les personnaliser en fonction des besoins spécifiques des clients. ECCO est un cadre conceptuel élaboré pour soutenir cette approche. L'outil automatise le processus de localisation

des parties réutilisables de produits, permettant ainsi la composition d'un nouveau produit en sélectionnant les caractéristiques nécessaires, offrant ainsi un moyen efficace de gérer la réutilisation de logiciels. L'outil Gears [KC14] repose sur des éléments uniques pour la modélisation des caractéristiques, les points de variation, et la configuration automatisée des produits. Il permet d'adapter efficacement les actifs d'ingénierie partagés à l'intérieur d'un portefeuille de produits, offrant une solution pour la gestion de la variabilité. Pure : : variant [Beu12] est un outil conçu pour simplifier le développement des LPL en gérant la complexité des éléments communs et des variations entre les différentes variantes de produits. Il offre des capacités de modélisation pour surmonter ces défis. Mobioos Forge [GZ22] est une plateforme conçue pour simplifier la gestion des personnalisations complexes, des configurations et de la réutilisation des applications. Son objectif principal est de réduire les coûts de développement et de maintenance tout en accélérant le déploiement des applications. Mobioos Forge s'intègre dans les flux de travail existants, garantissant ainsi la conformité avec les solutions préexistantes.

La gestion des LPL est un domaine en constante évolution, et les outils de migration jouent un rôle important pour faciliter ce processus tout en renforçant leur qualité et leur robustesse. Chacun des outils présentés apporte des caractéristiques spécifiques qui contribuent à la personnalisation, à la réutilisation, et à la gestion des configurations complexes d'applications. Elles mettent l'accent sur la personnalisation, la gestion de la variabilité, l'amélioration de la qualité des modèles et la correction des défauts. En combinant ces outils avec des stratégies de gestion appropriées, les développeurs peuvent efficacement gérer les défis liés aux LPL, réduire les coûts de développement et de maintenance, accélérer le déploiement des applications et garantir la conformité avec les exigences préexistantes. Dans nos travaux présentés dans cette thèse, nous proposons une méthode automatisée qui se sert des exigences en agilité pour produire des modèles de caractéristiques. Notre approche s'inscrit dans la lignée des frameworks existants pour la production de LPL. Nous ajoutons à ces approche l'intégration des méthodes agiles et de méthodes formelles.

2.4.3 Retours d'expérience

Les retours d'expérience en matière de migration des LPL fournissent des informations utiles sur les approches adoptées par différentes organisations pour gérer cette transition complexe. Dans cette section, nous explorerons trois retours d'expérience qui mettent en lumière les défis et les succès liés à la migration vers des LPL.

L'expérience de Salion [BCK03] est un exemple d'une approche réactive pour la migration vers des LPL. En adoptant cette approche, Salion a réussi à effectuer cette transition avec un effort moindre par rapport à la construction du logiciel de référence. Cette migration a permis à Salion de réduire les délais de mise sur le marché de ses produits. De nombreuses organisations commencent par réutiliser des logiciels en clonant des systèmes existants et en les adaptant aux besoins spécifiques des clients. Cependant, cette

approche peut entraîner des problèmes de synchronisation à mesure que le nombre de systèmes clonés augmente. Dans de tels cas, les organisations optent souvent pour la migration vers une LPL, ce qui permet de réduire les coûts de développement et de maintenance. Le retour d'expérience de l'extraction d'une LPL basée sur des préprocesseurs à partir de cinq jeux Android clonés, tels qu'Apo-Games [ÅNKB19], offre des informations essentielles pour mieux comprendre le processus d'adoption de LPL. Plusieurs enseignements sont dégagés de leur étude, notamment les défis introduits par les différences liées aux bibliothèques externes, affectant la cohérence des implémentations. L'effort pour personnaliser l'identité visuelle des jeux souligne la nécessité de repenser la conception des caractéristiques communes. La complexité de la prise de décision sur l'inclusion de certaines caractéristiques met en évidence la difficulté de déterminer celles qui doivent être considérées. Enfin, l'absence d'outils établis pour soutenir efficacement le processus d'extraction souligne le besoin de développer des outils plus robustes et adaptés aux besoins spécifiques de l'ingénierie de LPL. Il peut arriver qu'une organisation doive retirer et remplacer une LPL, même dans le cas d'une LPL qui devient obsolète. Le processus de retrait, défini dans des études de cas réelles [CnKL+23], prend en compte les technologies obsolètes, facilitant ainsi la transition vers le développement d'une nouvelle LPL.

Les retours d'expérience présentés ici démontrent la diversité des approches adoptées pour migrer vers des LPL. Qu'il s'agisse d'approches réactives, d'extraction de LPL à partir de systèmes clonés ou de processus de retrait et de remplacement, ces expériences fournissent des enseignements précieux pour les organisations qui cherchent à gérer cette transition complexe. Chaque expérience met en lumière les avantages de la migration vers des LPL, notamment la réduction des coûts de développement et la flexibilité accrue dans le cycle de vie du logiciel.

2.4.4 Gestion extractive de la variabilité

Les approches extractives se concentrent sur la gestion de la variabilité après la création de la LPL, et elles abordent divers aspects liés à la réutilisation de logiciels dans des familles de systèmes. Les LPL concernent les familles de systèmes qui partagent des ressources communes, permettant une réutilisation cadrée en localisant les caractéristiques. La localisation des caractéristiques est l'activité consistant à identifier un emplacement dans le code source qui permet la mise en œuvre des caractéristiques d'un système logiciel [DRGP13].

Les SPL commencent rarement à partir de rien. Elles débutent généralement à partir d'un ensemble de systèmes existants qui subissent un processus de réingénierie. Cependant, il existe encore des problèmes tels que la mise en œuvre de l'automatisation et du support des outils, l'utilisation de différentes sources d'information, la nécessité d'améliorations dans la gestion des caractéristiques, la définition de moyens pour combiner différentes stratégies et méthodes, le manque de *refactoring* sophistiqué et le besoin de nouvelles mesures plus robustes [ALL+17]. L'analyse de la variabilité des artefacts exis-

tants reflète différentes perspectives des parties prenantes et facilite les décisions dans l'adoption de la LPL [IRW16]. Compte tenu du fait que les exigences guident de nombreuses méthodes et activités de développement, il est intéressant d'analyser la variabilité des comportements tels qu'ils sont présentés dans les exigences fonctionnelles. L'adoption de l'ingénierie des LPL pour soutenir la réutilisation systématique d'artefacts liés au logiciel au sein des familles de produits est un défi complexe, chronophage et source d'erreurs [IRW16].

En conclusion, les approches extractives de migration vers les LPL offrent des perspectives variées pour gérer la variabilité dans les logiciels existants. Ces approches abordent des défis tels que la réingénierie, la localisation des caractéristiques, la migration pour surmonter la faible qualité des produits, l'analyse de la variabilité des exigences et la migration pour surmonter le clonage. Nos travaux s'inscrivent au sein des approches extractives. En effet, nous cherchons à identifier la variabilité au sein d'une famille de logiciels déjà développés.

2.4.5 Gestion proactive de la variabilité

Les approches proactives de gestion de la variabilité dans les LPL visent à intégrer dès le début du développement la planification et l'intégration des caractéristiques de la LPL [PBvdL05]. Cette approche prévoit une gestion anticipée de la variabilité, garantissant que les caractéristiques de la LPL sont préalablement planifiées et intégrées dès le départ du processus de développement.

De nombreux outils ont été développés pour mettre en œuvre des approches proactives dans les LPL. Parmi ces outils, on peut citer FeatureIDE [MTS+17], Gears [KC14], pure : : variant [Beu12], et ProDSPL [APAF21]. Ces outils facilitent la planification, la gestion et l'intégration anticipée des caractéristiques au sein des LPL, offrant ainsi des solutions pratiques pour la gestion de la variabilité dès le départ du développement. Les LPL dynamiques (DSPL) représentent une approche de gestion de la variabilité en temps d'exécution, permettant une auto-adaptation en fonction du contexte. Dans le contexte des DSPL, de nombreuses approches proactives appliquent des mesures dès qu'un changement de contexte se produit, permettant ainsi d'anticiper les variations et de réagir de manière précoce. L'outil ProDSPL [APAF21] est un exemple de solution proactive dans le domaine des DSPL. Il utilise un modèle du système pour anticiper les futures variations du système et génère la meilleure configuration DSPL pour minimiser l'impact négatif des événements futurs sur les exigences de qualité du système. Cette capacité de prédiction favorise les adaptations stables qui nécessitent moins de reconfigurations, contribuant ainsi à la stabilité du système.

Les approches proactives dans le domaine des LPL offrent une gestion précoce et anticipée de la variabilité, garantissant que les caractéristiques sont intégrées dès le début du développement. Les outils proactifs tels que FeatureIDE, Gears, pure : :variant et ProDSPL facilitent cette démarche en fournissant des solutions pratiques pour la gestion

proactive de la variabilité. De plus, dans le contexte des LPL dynamiques (DSPL), les approches proactives contribuent à anticiper et à réagir rapidement aux changements de contexte, renforçant ainsi la stabilité et la qualité du système. Nous nous sommes inspirés des résultats tirés de ces études pour contribuer à améliorer les pratiques de gestion de la variabilité dans le développement de logiciels basés sur des LPL.

2.4.6 Gestion réactive de la variabilité

Les approches réactives constituent une version itérative et adaptative des méthodes proactives de gestion des LPL. Dans cette section, nous explorons les pratiques réactives en matière de développement de LPL, en mettant l'accent sur la réactivité dans la gestion de la variabilité et les avantages qu'elle peut offrir.

Les organisations ont traditionnellement développé des variantes de produits en adoptant une approche de clonage et de personnalisation, c'est-à-dire en copiant et en adaptant des systèmes pour répondre à de nouvelles exigences [KB20]. Cependant, cette technique, bien que simple et largement disponible, comporte des inconvénients majeurs. Elle limite le passage à l'échelle du nombre de variantes et en cas de nécessité de faire évoluer ces variantes, elle peut exiger une coûteuse réingénierie pour les transformer en une plateforme logicielle configurable, également connue sous le nom de LPL. La gestion réactive de la variabilité s'aligne également avec le développement agile de logiciels. Les organisations agiles peuvent souhaiter se concentrer sur la satisfaction de leur base de clients actuelle sans s'encombrer des pratiques traditionnelles de gestion de la variabilité [GAM10]. Cependant, les méthodes agiles peuvent être adaptées pour gérer efficacement la variabilité. Les pratiques agiles, telles que le développement itératif, le *refactoring*, l'intégration continue et les tests continus, peuvent être exploitées pour gérer la variabilité au sein d'un projet de LPL. Avec la montée en puissance des plateformes de développement collaboratif basées sur des systèmes de gestion de versions distribués, la réutilisation de logiciels connaît une croissance significative. De nombreux développeurs créent des variantes en utilisant la technique du *forking* [BONB22], c'est-à-dire en créant une copie du code source existant pour répondre aux besoins de différents clients, marchés ou environnements. Ces variantes *forkées* deviennent ainsi une famille de logiciels partageant une base de code commune, et elles sont maintenues en parallèle par les mêmes ou différents développeurs. Il en résulte la création de familles de logiciels au sein des écosystèmes logiciels, caractérisés par de vastes collections de composants logiciels interdépendants entretenus par des communautés de contributeurs. Un exemple probant de l'efficacité des méthodes réactives est l'entreprise Salion Inc [BCK03]. Salion a adopté une approche réactive pour passer de produits logiciels individuels à une LPL, ce qui a nécessité considérablement moins d'efforts que la création d'un logiciel de référence. Cette transition a permis à Salion de réduire de manière significative ses délais de mise sur le marché. Plus précisément, Salion a réussi cette transition avec un effort équivalent à seulement 1% de ce qui aurait été nécessaire pour construire leur produit

logiciel de base. Cette démarche représente une réduction considérable par rapport aux efforts généralement requis pour une transition proactive vers des lignes de produits logiciels. IsiSPL [Hla22] est une méthode d'automatisation du processus d'ingénierie des LPL basée sur les stratégies d'adoption réactive [HSD21] ou extractive. Cette méthode propose l'identification d'artefacts à granularité fine, l'application d'une technique de localisation des caractéristiques basée sur l'analyse de concepts formels et relationnels, ainsi que la mise en œuvre d'une plateforme configurable par annotations et un procédé permettant d'atténuer les problèmes de lisibilité d'annotations dans le code.

Les approches réactives dans le développement de LPL offrent une approche souple et itérative pour gérer la variabilité et promouvoir la réutilisation de logiciels. En mettant l'accent sur l'adaptabilité et la réactivité, ces méthodes s'alignent bien avec les pratiques agiles et peuvent réduire les efforts nécessaires pour passer à une LPL, tout en améliorant la rapidité de mise sur le marché. La gestion réactive de la variabilité peut donc être un atout précieux pour les organisations cherchant à optimiser leur développement de produits logiciels. Nos travaux s'inscrivent à chaque itération du cycle de développement d'un nouveau logiciel afin d'avoir une gestion de la variabilité au fur et à mesure qu'elle se présente pendant le processus de développement. Nous proposons deux méthodes d'identification de la variabilité, l'une étant plus spécifique au cycle de développement de l'entreprise ITK.

La migration de logiciels vers des LPL représente un défi complexe, mais aussi une opportunité pour les organisations. Les différentes approches de migration, qu'elles soient proactives, extractives ou réactives, offrent des moyens variés pour gérer la variabilité dans le développement de logiciels. Ces approches permettent de réduire les coûts de développement, d'améliorer la qualité des produits logiciels, et d'offrir une plus grande flexibilité dans le cycle de vie du logiciel.

Les recherches présentées dans la continuation de ce manuscrit s'inscrivent comme une contribution aux approches extractives. En effet, nous étudions les pratiques de développement au sein de la famille de logiciels déjà élaborée par notre partenaire industriel, l'entreprise ITK. Notre objectif est d'analyser les caractéristiques des logiciels existants afin de proposer une gestion de la variabilité pour la conception de nouvelles solutions.

2.5 Analyse formelle de concepts pour la variabilité

L'analyse formelle de concepts et son extension, l'Analyse Relationnelle de Concepts, sont utilisées dans divers domaines de l'ingénierie logicielle et fréquemment appliquées à l'ingénierie des LPL. Ces méthodes formelles sont particulièrement adaptées aux traitements liés à la variabilité des attributs sur des objets, surtout lorsque ces objets sont des logiciels et que leurs attributs représentent des caractéristiques et des fonctionnalités.

2.5.1 analyse formelle de concepts

L'analyse formelle de concepts [GW99] (AFC) est un cadre mathématique basé sur la théorie des treillis, utilisé notamment pour analyser et organiser des données, et basé sur le concept de concepts formels. Elle permet de découvrir des relations entre les objets et les attributs au sein d'un ensemble de données. L'AFC est largement utilisée dans divers domaines [Pri06] tels que l'exploration de données, la recherche d'informations, la représentation des connaissances, l'ingénierie des ontologies et l'ingénierie logicielle [CDFR08, PIKD13].

Au cœur de l'AFC se trouve le concept de contexte formel. Un contexte formel est constitué d'un ensemble d'objets, d'un ensemble d'attributs et d'une relation binaire entre les objets et les attributs. La relation binaire indique si un objet possède ou non un certain attribut. En d'autres termes, elle établit une connexion entre les objets et les attributs en fonction de leur présence ou de leur absence. Un contexte formel peut être représenté sous la forme d'un tableau à deux dimensions, appelé matrice de contexte formel. Les lignes de la matrice représentent les objets, les colonnes représentent les attributs, et les entrées indiquent la présence ou l'absence des attributs pour chaque objet. Cette matrice sert de base à l'analyse ultérieure dans l'AFC. Le tableau 2.1 représente les

Table 2.1 – Contexte formel décrivant les diverses prédictions selon le type de culture

	météo	maladie	sol	phénologie	carbone
Vigne	x	x	x		
Banane			x	x	x
Tomate	x			x	x
Pomme		x	x	x	
Amande	x		x	x	
Blé	x	x			x
Cacao		x	x		x

caractéristiques des logiciels présentés dans la figure 2.1 où chaque logiciel associé à une culture a comme caractéristique la présence ou l'absence de certains types de prévisions.

En utilisant le contexte formel, l'AFC vise à identifier des concepts formels. Un concept formel est une paire constituée d'un ensemble d'objets et d'un ensemble d'attributs. Il représente un sous-ensemble d'objets *maximal* qui partagent un ensemble commun *maximal* d'attributs, et vice versa. Par exemple, sur la table 2.1, on peut observer que les logiciels dédiés aux cultures *Banane* et *Tomate* partagent *phénologie* et *carbone*. Ce sont les seuls logiciels à avoir les deux caractéristiques et ils ne partagent rien d'autre. $(\{Banane, Tomate\}, \{phénologie, carbone\})$ est donc un concept formel.
$(\{Vigne,Blé\}, \{météo\})$ n'est en revanche pas un concept formel. En effet :

- soit on peut compléter les logiciels partageant *météo* par *Tomate* et *Amande*, donnant le concept formel $(\{Vigne,Blé, Tomate, Amande\}, \{météo\})$,

— soit on peut compléter les caractéristiques communes à *Vigne* et *Blé* par *maladie*, donnant le concept formel ($\{$*Vigne*,*Blé*$\}$, $\{$*météo*, *maladie*$\}$).

Formellement, pour un contexte formel (O, A, R) où O est l'ensemble de tous les objets, A l'ensemble de tous les attributs et R la relation, un concept formel (E, I) est défini par : $E \subseteq O$, un sous-ensemble d'objets (extension) et $I \subseteq A$ un sous-ensemble d'attributs (intension). Si on définit les applications f et g qui associent respectivement les attributs communs à un ensemble d'objets (f) et les objets à un ensemble d'attributs lorsqu'ils les partagent (g), i.e. $f : \mathcal{P}(O) \to \mathcal{P}(A)$ and $g : \mathcal{P}(A) \to \mathcal{P}(O)$ avec $f(X) = \{y \in A | X \times \{y\} \subseteq R\}$ and $g(Y) = \{x \in O | \{x\} \times Y \subseteq R\}$, alors pour un concept (E, I), $f(E) = I$ et $g(I) = E$. Pour un concept C on notera E et I respectivement $Extent(C)$ et $Intent(C)$. Pour le concept formel ($\{$*Banane*, *Tomate*$\}$, $\{$*phénologie*, *carbone*$\}$), $\{$*Banane*, *Tomate*$\}$ est l'extension et $\{$*phénologie*, *carbone*$\}$ est l'intension.

Un concept formel peut être compris comme une généralisation ou une abstraction qui capture les caractéristiques communes partagées par un groupe d'objets. Il permet d'organiser et de classer les données en fonction de ces caractéristiques communes.

Le treillis de concepts et l'AOC-Poset L'ensemble de tous les concepts formels au sein d'un contexte formel donné peut être structuré en un ensemble partiellement ordonné, appelé Treillis de concepts [Pet01]. L'ordre entre concepts est donné par la relation d'inclusion entre extensions et intensions. Formellement, (E_{sub}, I_{sub}) est un sous-concept de (E_{sup}, I_{sup}) lorsque $E_{sub} \subseteq E_{sup}$. Dans ce cas, de manière équivalente, $I_{sup} \subseteq I_{sub}$. Le concept ($\{$*Vigne*, *Blé*, *Tomate*, *Amande*$\}$, $\{$*météo*$\}$) (numéroté 1 sur la figure 2.3) est ainsi un super-concept du concept ($\{$*Vigne*, *Blé*$\}$, $\{$*météo*, *maladie*$\}$) (numéroté 2 sur la figure 2.3). Extensions et intensions varient ainsi en sens inverse : un concept est positionné plus haut dans le treillis s'il englobe un ensemble plus vaste d'objets et un ensemble plus petit d'attributs que ses sous-concepts. La figure 2.3 présente le treillis de concepts associé au contexte formel de la table 2.1. Le concept 21 groupe *Tomate* et *Banane* qui partagent *carbone* et *phénologie*.

L'AOC-Poset (Attribute-Object Concept partially ordered set) [BGH+14] est un sous-ordre spécifique du treillis de concepts, restreint aux concepts introducteurs d'au moins un objet ou un attribut. Un concept introducteur d'attribut est le concept le plus grand possédant l'attribut dans son extension et ses sous-concepts en héritent. Un concept introducteur d'objet est le concept le plus petit possédant l'objet dans son extension et ses super-concepts en héritent. La figure 2.4 présente l'AOC-poset associé au contexte formel de la table 2.1.

Lorsqu'on visualise un treillis ou un AOC-Poset [DLBH13], on utilise un diagramme de Hasse. Chaque nœud dans ce diagramme représente un concept formel, et les arêtes indiquent la relation d'inclusion entre ces concepts. Contrairement à une représentation exhaustive, les relations de transitivité ne sont pas explicitement représentées, ce qui simplifie la visualisation tout en préservant la structure d'ordre. Cette représenta-

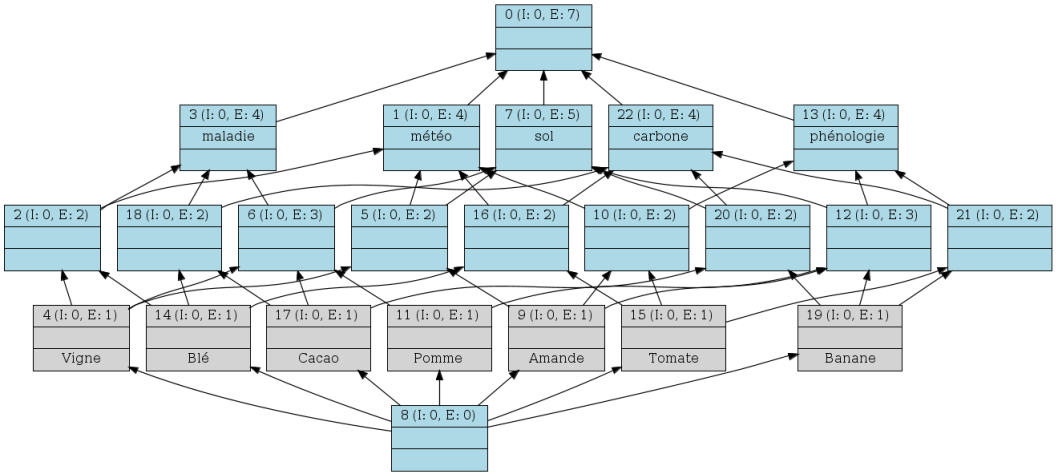


FIGURE 2.3 – Treillis de concepts construit d'après le contexte formel de la table 2.1 avec l'outil Cogui (https ://www.lirmm.fr/cogui/). Chaque étiquette de concept est constituée dans les sorties de Cogui en trois parties. Dans la partie supérieure de chaque étiquette de concept, on lit le numéro du concept, puis entre parenthèses derrière le symbole `I :` le cardinal de l'intension, et derrière le symbole `E :` le cardinal de l'extension. La partie centrale indique les attributs introduits. La partie inférieure indique les objets introduits.

tion graphique offre une vue systématique des relations et des dépendances entre les concepts au sein du contexte formel, facilitant ainsi l'exploration et la compréhension de la structure conceptuelle sous-jacente. En plus de la représentation sous forme de diagramme de Hasse, une vue simplifiée est donnée sur les concepts : seuls les attributs et les objets introduits par un concept sont montrés.

En résumé, l'AFC est un cadre mathématique qui utilise des contextes formels pour analyser et organiser des données. Les contextes formels sont constitués d'objets, d'attributs et d'une relation binaire entre eux. L'AFC vise à identifier des concepts formels, chacun étant une paire d'ensembles maximaux représentant respectivement un sous-ensemble d'objets et le sous-ensemble des attributs qu'ils partagent. L'ensemble de tous les concepts formels peut être organisé en un ensemble partiellement ordonné, le treillis de concepts qui représente la hiérarchie des concepts basée sur leurs relations d'inclusion. Un sous-ordre particulier du treillis, appelé AOC-Poset ou poset AOC, nous intéressera particulièrement dans ce travail car il est de plus petite taille que le treillis et on peut y lire les relations logiques qui nous intéressent.

2.5.2 Analyse relationnelle de Concepts

L'analyse relationnelle de concepts (ARC), [HHNV13] est une méthode basée sur la théorie des treillis pour regrouper des objets décrits par des liens inter-objets et par des

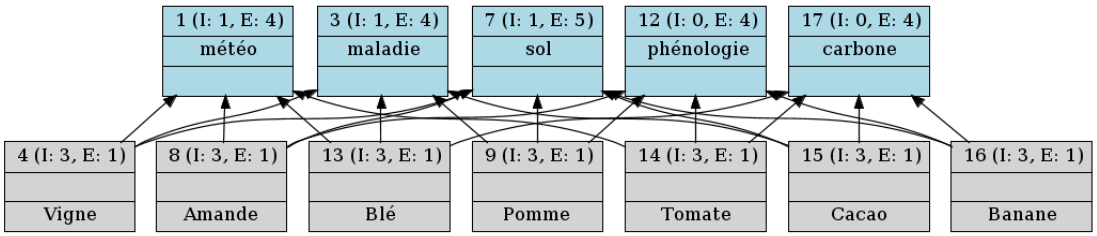


FIGURE 2.4 – AOC-Poset associé au contexte formel de la table 2.1 construit avec l'outil Cogui. Les concepts du treillis de la figure 2.3 qui n'introduisent aucun objet ni aucun attribut, comme le concept étiqueté 2, n'apparaissent plus dans l'AOC-Poset. Dans cet exemple, l'AOC-Poset a une forme particulière : tous les attributs sont introduits dans la couche haute de la structure et tous les objets dans une couche basse. Il ne s'agit pas d'un cas général comme les exemples réels le montreront dans la suite du manuscrit.

caractéristiques primitives. C'est une extension de l'AFC qui permet d'aller au-delà de la description des objets par les seules caractéristiques primitives. L'ARC est un moyen pour la représentation de la variabilité au sein d'une famille de contextes là où l'AFC la représente pour un unique contexte.

L'ARC prend en entrée un ensemble de *contextes formels* qui sont des tableaux décrivant les objets (ou éléments) par leurs caractéristiques primitives, et des *contextes relationnels* décrivant les relations entre les éléments d'un contexte formel s'il s'agit d'une relation interne ou de deux contextes formels sinon. L'ARC permet de prendre en compte ces contextes formels et relationnels, d'y former des groupes d'objets et d'en extraire des relations logiques. Les tableaux 2.2 à 2.5 décrivent une telle famille de contextes. Le contexte formel des logiciels les décrit dans le tableau 2.2, en utilisant uniquement un identifiant (qui est leur propre nom), ce qui donne une matrice diagonale. Les prédictions sont décrites dans le contexte formel du tableau 2.3, sans attribut descriptif. Le tableau 2.4 est un contexte relationnel qui indique quel logiciel dispose de quelle prédiction. D'une certaine manière il re-transcrit les informations du tableau 2.1 utilisé pour décrire l'AFC. Enfin, le tableau 2.5 représente les liens d'*influence* entre les différents types de prédictions. Quatre liens ont été identifiés. Le premier concerne l'influence de la météo sur la propagation des maladies. Ensuite, la qualité du sol influe sur la phénologie. Le lien entre la phénologie et le carbone examine comment la croissance des cultures affecte le stockage de carbone. Enfin, la météo influence la capacité du sol à stocker le carbone.

L'ARC est un cadre permettant, comme l'AFC, de construire des structures conceptuelles (hiérarchies de concepts telles que des treillis de concepts ou des AOC-Posets), des implications (règles entre attributs d'un contexte), ou tout autre type de relation logique. Chaque concept est un groupe maximal d'objets partageant une description maximale, composée de leurs caractéristiques primitives communes comme pour l'AFC, auxquelles s'ajoutent leurs relations communes avec d'autres concepts, capturées par des attributs

relationnels. Un attribut relationnel est constitué d'un opérateur logique (par exemple *exist*, *forall*, *contains* et des variations avec des pourcentages), d'une relation (nom d'un contexte relationnel) et d'un concept cible de l'attribut. Par exemple, si un logiciel dispose de l'attribut relationnel *exist hasPrediction(Concept_Prediction_0)*, cela signifie qu'il a *au moins un lien par la relation hasPrediction vers une des prédictions de l'extension de Concept_Prediction_0*.

La figure 2.5 présente ces structures construites à partir des tableaux 2.2 à et2.5. Deux AOC-Posets ont été construits, l'un pour les logiciels (à gauche) et l'autre pour les prédictions (à droite).

Les prédictions n'ayant pas d'attribut primitif, elles sont structurées par leurs liens d'influence. Par exemple, le *Concept_Prediction_1* groupe des prédictions (*météo* et *phénologie* introduits dans ce concept et *sol* hérité du sous-concept) qui influencent au moins l'une des prédictions du *Concept_Prediction_0* (qui groupe toutes les prédictions : *maladie* et *carbone* introduits dans ce concept et qui n'influencent rien, ainsi que les prédictions héritées des sous-concepts). Ceci est capturé dans l'attribut relationnel *exist influence(Concept_Prediction_0)* introduit dans *Concept_Prediction_1*. En effet, *météo*, *phénologie* et *sol* influencent chacune au moins une autre prédiction. L'attribut relationnel *exist influence(Concept_Prediction_1)* que possède *sol* traduit le fait que *sol* influence au moins l'une des prédictions de l'extension du *Concept_Prediction_1* ce qui est le cas car il influence *phénologie*.

L'AOC-Poset des logiciels les structure suivant leurs identifiants et leurs liens avec les prédictions. Par exemple, *Vigne, Banane, Pomme, Amande, Cacao* ont tous comme prédiction *sol* (introduit dans *Concept_Prediction_2*), ce qu'indique l'attribut relationnel partagé *exist hasPrediction(Concept_Prediction_2)* introduit dans *Concept_Software_8* et hérité par ces logiciels.

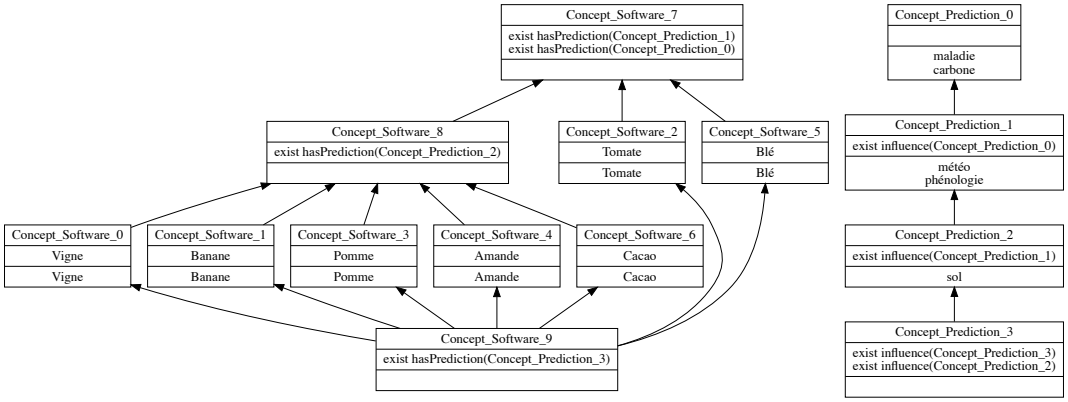


FIGURE 2.5 – Exemple d'application de l'ARC avec des AOC-Posets sur les logiciels de cultures et les prédictions. La famille de contextes utilisée est constituée des tableaux 2.2, 2.3, 2.4 et2.5

TABLE 2.2 – Contexte formel Logiciel (software)

	Vigne	Banane	Tomate	Pomme	Amande	Blé	Cacao
Vigne	x						
Banane		x					
Tomate			x				
Pomme				x			
Amande					x		
Blé						x	
Cacao							x

TABLE 2.3 – Contexte formel Prédiction (Prediction)

météo
maladie
sol
phénologie
carbone

Table 2.4 – Contexte relationnel : quels logiciels ont quelles prédictions (hasPrediction)

	météo	maladie	sol	phénologie	carbone
Vigne	x	x	x		
Banane			x	x	x
Tomate	x			x	x
Pomme		x	x	x	
Amande	x		x	x	
Blé	x	x			x
Cacao		x	x		x

Table 2.5 – Contexte relationnel : influences mutuelles des types de prédiction (influence)

	météo	maladie	sol	phénologie	carbone
météo		x			x
maladie					
sol				x	
phénologie					x
carbone					

Nous renvoyons le lecteur à la référence [HHNV13] pour la formalisation de l'ARC.

2.5.3 Application de l'analyse formelle de concepts à l'ingénierie logicielle

L'ingénierie logicielle est un domaine en constante évolution, où les méthodes et les outils jouent un rôle dans la conception, le développement, la maintenance, et la compréhension des systèmes logiciels. L'AFC est une approche mathématique qui trouve des applications dans divers domaines de l'ingénierie logicielle [TCBE05]. Elle offre des outils pour explorer les relations sémantiques entre les éléments du logiciel, permettant ainsi de résoudre des problèmes complexes.

L'AFC peut être appliquée dans plusieurs domaines de l'ingénierie logicielle et nous pouvons en citer ici quelques-uns. L'un de ces domaines est la vérification automatique des propriétés des systèmes embarqués en temps réel critiques pour la sécurité. L'analyse statique par interprétation abstraite, basée sur l'AFC, est efficace pour cette application [Cou96]. L'AFC est utilisée pour la compréhension des classes dans la maintenance des systèmes orientés objet, offrant une vue des relations internes [ADN03]. Elle est également employée pour la migration de systèmes vers l'orienté objet [SMLD97], la

modularisation du code patrimonial *(legacy)* [LS97], l'identification de modules dans du code patrimonial [SR99], la localisation de défauts [CDFR08] et la génération de règles de transformation de modèles [SDH+12]. De plus, l'AFC est utilisée pour la maintenance des hiérarchies de classes [GM93], l'extraction de parties génériques à partir de modèles logiciels [DHH+04] et la reconstruction de correspondances entre caractéristiques et code source [EKS03]. Elle trouve également des applications dans la localisation de concepts dans le code source [PGM12] et la classification de composants pour la composition de services Web [AAF+09]. L'ARC, une extension de l'AFC, est utilisée par exemple pour la sélection de services Web en fonction des besoins de l'utilisateur [ADH+11].

Pour résumer l'AFC, est une approche mathématique pour résoudre des problèmes complexes dans divers domaines de l'ingénierie logicielle. Ses applications vont de la vérification des propriétés des systèmes critiques à la maintenance des hiérarchies de classes, en passant par la ré-ingénierie des systèmes patrimoniaux et la recherche d'informations dans le code source. L'utilisation de l'ARC étend encore les possibilités de l'AFC, notamment dans la composition de services Web et plus généralement pour l'analyse d'artefacts logiciels liés entre eux. Les recherches dans ces domaines continuent d'évoluer, ouvrant de nouvelles perspectives pour l'ingénierie logicielle. Dans notre contexte, l'AFC et l'ARC s'intègrent à nos méthodes de réingénierie des systèmes existants. Les travaux décrits dans cette thèse reposent sur les avancées en ingénierie logicielle et sont appliqués à l'analyse des logiciels développés par notre partenaire, l'entreprise ITK.

2.5.4 Application de l'analyse formelle de concepts aux LPL

La gestion de la variabilité est nécessaire dans l'ingénierie des LPL. Elle permet de cadrer la prolifération des caractéristiques. Pour résoudre ce problème, l'AFC est une approche ayant prouvé son efficacité sur plusieurs sujets.

L'AFC est employée pour la restructuration de la variabilité, permettant de classer les caractéristiques variables et de définir des stratégies de restructuration. Par exemple, Loesch et Ploedereder [LP07a] ont proposé une méthode basée sur l'AFC pour analyser la variabilité dans les LPL, en construisant un treillis qui fournit une classification des relations logiques décrivant l'utilisation des caractéristiques variables. Cette classification a permis de cibler les caractéristiques pouvant être fusionnées pour simplifier la variabilité. De plus, elle a mis en lumière les caractéristiques spécifiques à certaines configurations de produits, indiquant celles qui pourraient être éliminées si elles ne sont pas destinées à être utilisées dans d'autres configurations à l'avenir. La modélisation de la variabilité est essentielle pour documenter les caractéristiques communes et variables dans les LPL. Les treillis de concepts offrent une vision double des caractéristiques et des configurations, offrant des avantages pour la gestion de la variabilité. En particulier, le treillis conceptuel se distingue en tant que représentation canonique, fournissant une structure graphique qui facilite la compréhension et l'analyse approfondie des configurations logicielles [CBHN20]. La migration vers des logiciels orientés carac-

téristiques peut être accélérée en générant automatiquement des modèles à partir de variantes de produits existants. Certaines méthodes, basées sur l'AFC, examinent des matrices d'incidence contenant des relations de correspondance et produisent des modèles de caractéristiques précis. L'une des premières se trouve décrite dans [RPK11]. L'AFC est utilisée pour extraire des modèles de caractéristiques à partir des exigences fonctionnelles des variantes de produits. Cela permet d'identifier les caractéristiques potentielles de chaque variante de produit et de distinguer les caractéristiques obligatoires des facultatives [MBB16]. Galasso et al. [CHN19c] ont également étudié l'identification de la variabilité à l'aide de l'AFC. Ils ont proposé une méthode d'extraction des caractéristiques, des attributs à valeurs multiples et des cardinalités, sous forme de relations logiques. Dans le domaine de la réingénierie des lignes de produits, la comparaison de variantes de diagrammes de cas d'utilisation est utilisée pour détecter les éléments communs et variables. Une approche proposée est d'utiliser l'AFC pour visualiser les caractéristiques communes et variables [ABS+22]. Hlad et al. [HLHS21] proposent une approche de localisation de caractéristiques reposant sur l'AFC et l'ARC, afin de pallier les limitations observées dans certaines applications de l'AFC, notamment en termes de précision des traces, d'interaction entre caractéristiques et de granularité des artefacts. Dans ce contexte, ils suggèrent une extension des techniques de localisation de caractéristiques basées sur l'AFC et l'ARC. Leur méthode extrait des traces précises des caractéristiques et de leurs interactions. Les résultats obtenus montrent que cette approche peut non seulement récupérer les liens entre les artefacts et les caractéristiques, mais aussi réduire le nombre de caractéristiques impliquées, rendant ainsi les traces, et les annotations qui en découlent, plus compréhensibles.

En résumé l'AFC s'avère être un outil utilisé dans différents domaines de l'ingénierie des LPL. Elle permet de gérer efficacement la variabilité, d'accélérer la migration vers des logiciels orientés caractéristiques, de modéliser la variabilité, d'extraire des modèles de caractéristiques et de comparer les variantes. Ces applications variées démontrent l'importance de l'AFC dans l'amélioration de la gestion de la variabilité et la réingénierie des LPL. L'extension de l'AFC, l'ARC, permet d'aller au-delà de la description des objets par des caractéristiques primitives en prenant en compte les relations inter-objets. Elle offre des possibilités encore plus étendues pour la modélisation de la variabilité et la construction de structures conceptuelles.

2.6 Conclusion

Thomas *rappel des conclusions* Dans cette section de l'état de l'art, nous avons présenté diverses approches, méthodologies et outils pour situer nos travaux par rapport à la littérature existante. L'ingénierie des LPL s'étend sur un vaste éventail de domaines de la représentation des connaissances aux méthodes formelles. Cette diversité de domaines ouvre des possibilités d'adaptation en fonction des contextes spécifiques. L'intégration

de l'agilité dans les LPL offre une adaptabilité importante, et la synergie entre ces deux approches s'avère particulièrement pertinente. La combinaison de l'agilité avec les méthodes formelles pour l'analyse du code présente les avantages de la flexibilité et de la rigueur. L'adoption d'une nouvelle technologie comporte des risques, et il est essentiel de comprendre l'état d'esprit préalable à une migration. Cela revêt une importance particulière pour les LPL, car elles modifient considérablement les pratiques, nécessitant ainsi une préparation rigoureuse. La migration constitue la dernière étape permettant d'utiliser une LPL tout en prenant en compte l'existant, que ce soit dans les pratiques ou dans le code lui-même. C'est en prenant en considération ces trois aspects que la suite de nos travaux a été menée : l'agilité pour favoriser la flexibilité, l'adoption pour comprendre l'existant et les perceptions, puis la migration pour concrétiser l'utilisation des LPL. Trois contributions émanant du contexte et des travaux existants ont été formulées. Tout d'abord, une évaluation et une analyse de la perception du projet ont été entreprises afin de formuler des recommandations propices à la réussite du projet. Ensuite, une identification de la variabilité a été réalisée à partir des schémas de données d'E/S des simulateurs. Enfin, l'exploitation des récits utilisateurs et des systèmes de contrôle de versions a été étudié pour représenter la variabilité au sein d'une famille de logiciels, tirant ainsi parti de l'agilité et des méthodes de développement actuelles.

CHAPITRE 3

Évaluation empirique de la perception de l'ingénierie de la gamme de produits logiciels par une PME avant la migration de sa base de code

Sommaire

La migration d'une gamme de logiciels similaires vers une ligne de produits logiciels est un processus coûteux et potentiellement préoccupant. En effet, l'ingénierie des lignes de produits logiciels peut avoir un impact significatif sur le processus de développement de l'entreprise et son adoption implique un changement dans les habitudes des développeurs. Les travaux présentés dans ce chapitre sont issus d'entretiens réalisés avec les salariés de notre partenaire industriel ITK. Nous avons procédé avec eux à une évaluation du processus de développement actuel et des pratiques au sein de l'entreprise. Nous avons également évalué les bénéfices et les risques attendus de cette migration. Des acteurs importants du projet ont été impliqués dans cette évaluation afin d'évaluer leur perception de cette migration et leur résistance au changement. Ce chapitre présente la conception des entretiens menés avec ces acteurs et une analyse de leurs résultats. Parmi les conclusions qualitatives de cette étude, nous avons observé que tous les acteurs, indépendamment de leur rôle dans le processus de développement, ont imaginé des avantages liés à leur propre activité dans cette migration. De plus, il a été observé qu'une stratégie efficace pour atténuer les risques implique de maintenir les parties prenantes conscientes du processus, ainsi que de conserver autant de bonnes pratiques que possible, tout en essayant de les faire participer au processus de migration pour assurer une transition en douceur et éviter d'éventuelles difficultés.

3.1 Introduction

Une ligne de produits logiciels (LPL) peut être extraite d'une base de code dans laquelle est déjà construit un ensemble de systèmes logiciels partageant des caractéristiques communes afin de personnaliser rapidement et facilement un nouveau logiciel [FLLE14, MZB+15, MAZ17]. Cette approche d'extraction pour migrer la base de code vers une LPL est un processus complexe et chronophage. De plus, cela peut avoir un impact significatif sur les processus de développement déjà bien établis et pratiqués par l'organisation qui gère la base de code. Dans ce chapitre nous présentons les travaux préliminaires à l'adoption d'une approche extractive.

La base de code de l'entreprise partenaire ITK est organisée autour d'une plate-forme

appelée Cross, un framework dans lequel la réutilisation est une pratique bien établie.

Afin d'accompagner les développeurs de cette entreprise, nous avons prévu une transition en douceur vers un changement progressif dans l'organisation de leur base de code. Ce chapitre rend compte de la première étape de cette transition. Nous avons réalisé une évaluation des pratiques actuelles de l'entreprise et de sa perception de l'ingénierie des LPL. Cette évaluation a été mise en œuvre au moyen d'une série d'entretiens menés auprès d'acteurs du projet travaillant pour l'entreprise. L'objectif de ces entretiens est d'étudier la perception de la transition et de fournir des recommandations pour faciliter le processus de migration. L'objectif de ce chapitre est de présenter les conclusions de cette étude.

Dans ce chapitre, nous présentons la conception détaillée des entretiens, suivant les recommandations de Kitchenham et al. [KP08], puis nous exposons les résultats obtenus. Nos entretiens ont été conçus avec des variantes pour les différents types d'acteurs ayant participé. Ils ont été menés auprès d'un groupe ciblé de participants, représentant différents rôles au sein de l'organisation. Plus précisément, 15% du personnel d'ITK, dont 50% sont des développeurs, 30% des responsables de produits (PO) et 10% des agronomes, ainsi que tous les concepteurs UI/UX. Cette composition a couvert presque tous les intervenants impliqués dans le projet. Le reste du personnel était soit engagé dans d'autres projets, tels que l'élevage ou la séquestration du carbone, soit faisait partie de l'équipe d'administration, comprenant notamment les ressources humaines et la comptabilité. Ainsi, ils ne seront pas directement concernés par la migration.

Nous avons également abordé différentes perceptions possibles : celles des bonnes pratiques actuelles, ainsi que celles des avantages, des risques et des préoccupations liées à l'adoption d'une LPL.

Les résultats ont montré que la plupart des acteurs étaient favorables à l'idée de migrer leur base de code vers une LPL et demandaient une participation active à ce processus. Les développeurs étaient les acteurs les plus enthousiastes. Ils apprécient le défi technique et managérial. Les autres acteurs ont exprimé davantage de risques et de craintes, notamment concernant une maintenance plus complexe de la LPL et certaines menaces pesant sur la créativité dans leur travail quotidien d'UI/UX, entre autres. Les acteurs responsables de l'interface entre les clients et les développeurs sont les plus proches de l'activité commerciale et sont ceux qui avaient le plus de craintes.

Les résultats de ces entretiens ont été instructifs et l'interprétation de ces résultats a permis de définir des lignes directrices dans le processus de migration de la base de code d'ITK. Ils fournissent également des indices pour l'adoption d'un processus LPL agile, qui atténuera les risques et les craintes exposés.

Les trois principaux composants de notre étude empirique [1] sont les suivants :

— *Définition des objectifs* (Section 3.2) : Construire les entretiens dans un objectif clair. L'objectif principal de notre étude était l'évaluation de la perception de la migra-

1. Voir [SA04] au sujet de l'évaluation empirique d'un processus agile.

tion vers une LPL. Nous avons identifié quatre sujets sur lesquels les entretiens devaient porter : les bonnes pratiques actuelles, les avantages de la migration, les risques de la migration et les craintes.

— *Collecte de données* (Sections 3.3 et 3.4) : Construire un protocole fiable, i.e. la collecte de données doit être planifiée correctement. Dans notre étude, la collecte de données a été réalisée grâce aux entretiens conçus selon les quatre sujets mentionnés précédemment.

— *Interprétation des données* (Sections 3.5 et 3.6) : Anticiper et éviter les biais, i.e. l'interprétation des données doit être menée de la manière la moins partisane possible (en faveur des LPL). Nous avons engagé de nombreuses discussions approfondies pendant nos entretiens, en écoutant attentivement les préoccupations des personnes interrogées et en transcrivant leurs réponses qui sont partagées publiquement, après les avoir anonymisées, dans le dépôt suivant : `https://gite.lirmm.fr/tgeorges/interviewartefact`.

Les principales contributions de ce chapitre sont une méthodologie de création de questionnaires, une méthodologie pour la conduite d'entretiens, une méthode d'analyse des entretiens, et des recommandations pour maximiser le succès d'un tel projet.

Le reste du chapitre est organisé comme suit. La section 3.2 détaille le contexte et les questions de recherche. La section 3.3 décrit la méthodologie. La section 3.4 présente les résultats. Sur la base des résultats, nous présentons, dans la section 3.5, des orientations pour préparer la migration vers une LPL. Les menaces pour la validité sont discutées dans la section 3.6. La section 3.7 expose les travaux connexes. Enfin, la section 3.8 conclut notre article en résumant les principales conclusions et les travaux futurs.

3.2 Contexte et questions de recherche

La section 3.2.1 présente le contexte de cette évaluation empirique, tandis que la section 3.2.2 introduit les quatre questions de recherche autour desquelles elle s'articule.

3.2.1 Contexte de ce travail

ITK développe ses systèmes de manière agile avec un processus de développement et de livraison bien établi. Les applications développées aident les agriculteurs dans la prise de décision en leur permettant de prédire les maladies et les rendements des cultures pour leurs plantes et leur bétail. Le cœur de ces applications est un ensemble de simulateurs construits par les agronomes. Ensuite, les équipes informatiques développent les applications qui collectent des données à partir de dispositifs IoT ou de services en ligne afin d'alimenter les simulateurs et d'afficher les données de sortie dans des tableaux de bord. Le back-end est construit en Kotlin, tandis que le front-end est développé avec Angular,

en utilisant principalement Typescript. Le back-end se base sur Spring Boot, et la base de données est hébergée sur un serveur PostgreSQL. Les agronomes utilisent Python ou MATLAB pour développer les simulateurs.

L'équipe chargée du développement utilise une plateforme centrale, appelée Cross qui est leur framework pour construire le code des applications. Initialement, la plateforme était conçue pour générer des preuves de concept *(PoC)*. Au fil du temps, cet objectif a évolué et la plateforme s'est transformée en base pour les produits finaux. Il existe actuellement 7 variants dans la famille pour 3 PO. Les nouveaux variants sont développés selon les besoins spécifiques des clients, un nouveau variant étant introduit environ une fois par an. Il est possible de placer l'architecture et l'organisation d'ITK sur le *framework FEF* : Les niveaux de l'*Architecture*, tels qu'expliqués dans le *Framework FEF* introduit par [vdLSR07], traitent des moyens techniques pour construire un logiciel et déterminent la réalisation technique des produits au sein de la ligne de produit logiciel. Au Niveau 1, les architectures se limitent à des systèmes individuels, sans aucune focalisation visible sur la réutilisation. En passant au Niveau 2, il y a un virage vers la réutilisation d'infrastructures tierces, avec des composants logiciels communs définis tels que les *middleware*, mais sans incorporation d'artefacts spécifiques au domaine. En progressant vers le Niveau 3, les parties communes du domaine sont capturées et implémentées dans une plateforme logicielle, offrant une architecture de référence pour toutes les applications, principalement centrée autour de l'utilisation de la plateforme. Cependant, cette plateforme configurable manque de support pour la gestion de la variabilité. En avançant vers le Niveau 4, à la fois les parties communes du domaine et la variabilité sont capturées de manière systématique, et une architecture de référence est spécifiée pour couvrir l'ensemble de la ligne de produits logiciels. Les artefacts du domaine sont conçus pour prendre en charge la dérivation des produits, en accordant une attention à l'explicitation de la gestion de la variabilité au sein de l'architecture de la ligne de produits logiciels. Enfin, au Niveau 5, l'architecture de référence prend un rôle dominant, les architectures des nouvelles applications ne divergent que marginalement de celle-ci. La dérivation automatisée des produits devient réalisable grâce à l'utilisation de scripts, d'outils et de langages de haut niveau, tandis que le développement d'applications consiste principalement en la configuration au sein des limites de l'architecture de référence, ouvrant la voie à la possibilité de la configuration automatique des produits.

La plateforme d'ITK a entre 5 et 6 ans. Selon les critères du *Framework FEF* [vdLSR07] rappelés ci-dessus, la plateforme interne développée par ITK correspond à un *Niveau 3 de l'architecture : Plateforme logicielle* avec une architecture de référence commune à tous les produits. L'architecture actuelle contient un ensemble d'artefacts communs et des points de variation explicites sont définis pour les nouvelles variantes. L'objectif de la migration est d'atteindre un niveau d'architecture où les parties communes et la variabilité du domaine sont capturées, et une architecture de référence est spécifiée pour l'ensemble de la ligne de produits logiciels. Cela inclut la réutilisation

systématique basée sur un référentiel d'artefacts, une variabilité explicite dans les artefacts, et une architecture de référence explicite qui régit les variations admissibles dans les architectures d'application. La gestion de la variabilité est traitée explicitement dans l'architecture de la ligne de produits logiciels, déterminant les configurations autorisées pour les architectures d'application et définissant des points de variation et des contraintes pour la plupart de ces variations, y compris les règles auxquelles les variantes spécifiques à l'application doivent se conformer. L'objectif ultime est de fournir une configuration logicielle pour une synthèse automatisée des variants.

Le framework FEF comprend plusieurs catégories avec entre autres les niveaux d'architecture vus précédemment et les niveaux organisationnels. Selon le framework FEF [vdLSR07], au niveau *organisationnel*, ITK fonctionne au quatrième niveau. Cela implique la synchronisation et la coordination de l'ingénierie de domaine et de l'ingénierie d'application. Le développement du projet est réalisé grâce à une forte collaboration entre les agronomes et les développeurs. La migration de la base de code et la collaboration avec notre équipe de recherche ont été décidées lors d'une discussion collégiale avec les parties prenantes du projet. Avant de commencer la conception des entretiens, nous n'avions pas encore établi une méthodologie de recherche bien définie pour le reste du projet. Nous voulions d'abord évaluer les perceptions des acteurs de l'entreprise concernant la migration vers les LPL. Selon les résultats des entretiens, nous élaborerons une méthodologie concrète pour le projet de migration. La méthodologie visent à maximiser la réutilisation du code et à accélérer davantage la livraison de nouvelles applications. L'objectif est de migrer la base de code vers une LPL, à un niveau d'architecture supérieur (Niveau 4 ou 5 dans [vdLSR07]). Afin de faciliter cette transition en douceur, nous avons décidé de mener d'abord des entretiens avec les collaborateurs de l'entreprise pour évaluer leur perception de l'ingénierie des LPL. Les entretiens ont été conçus de manière à répondre aux questions de recherche présentées dans la section suivante.

Les prochaines phases de migration impliqueront une analyse de la variabilité, en commençant par établir un processus pour la localisation des caractéristiques afin de synthétiser des modèles de caractéristiques. Ensuite, l'étude des interactions et des dépendances entre les caractéristiques permettra d'obtenir une description complète du système.

3.2.2 Questions de recherche pour la conception des entretiens

Nous avons élaboré et réalisé les entretiens afin d'identifier la perception des acteurs de l'entreprise quant aux pratiques actuelles, aux avantages, aux risques et aux craintes avant la migration vers une Ligne de Produits Logiciels.

QR1 Quelle est la perception des acteurs de l'entreprise concernant les bonnes pratiques de développement actuelles ? Avec cette question, nous souhaitons mettre

en évidence la perception des bonnes habitudes actuelles des acteurs de l'entreprise, afin de les confronter aux pratiques proposées par les nouveaux pratiques de développement après la migration.

QR2 Quelle est la perception des acteurs de l'entreprise concernant les avantages de la future LPL ? L'idée ici est de discuter avec les acteurs de l'entreprise de la valeur ajoutée de la future LPL du point de vue du développement d'applications.

QR3 Quelle est la perception des acteurs de l'entreprise concernant les risques de la future LPL ? Une fois les avantages discutés, nous voulons ici comprendre les risques perçus par les acteurs au sujet de la présence de la future LPL dans leur travail quotidien.

QR4 Quelles sont les craintes des acteurs de l'entreprise induites par les risques précédemment identifiés ? Cette question permet d'évaluer les craintes ressenties par les acteurs de l'entreprise quant aux conséquences du projet de migration.

Les craintes font référence aux appréhensions ou aux préoccupations que les acteurs de l'entreprise peuvent avoir concernant le processus de transition. De telles craintes peuvent découler de l'incertitude quant au résultat, des défis potentiels lors du processus de migration et de la nécessité d'une planification adéquate et de stratégies d'atténuation pour faire face efficacement aux risques identifiés.

Nous avons élaboré une méthodologie qui a guidé la conception et la conduite ultérieure des entretiens, afin de répondre à ces questions.

3.3 Méthodologie

Les entretiens sont l'une des techniques directes de collecte de données permettant de mener des études sur le terrain et de recueillir des opinions [SSL08]. Nous les avons choisis plutôt qu'un questionnaire écrit ou un groupe de discussion. Comparés à un groupe de discussion, les entretiens individuels évitent que les répondants ne s'influencent mutuellement. Comparés à un questionnaire écrit, les entretiens exigent plus d'engagement et de temps de la part des répondants, car ils nécessitent des rencontres avec la personne qui fait passer les entretiens. En retour, ils sont interactifs et flexibles, car les questions peuvent être clarifiées en cas d'ambiguïté et la discussion peut être prolongée au-delà du jeu initial de questions. Nous expliquons dans cette section comment nous avons choisi les participants, puis comment nous avons conçu et réalisé les entretiens.

Notre évaluation empirique suit les principes exposés dans "Empirical Research Methods in Software Engineering" [WHH03]. La *définition des objectifs* consiste à formuler des objectifs de recherche bien définis, ce qui est nécessaire pour mener des études empiriques efficaces. Définir l'objectif de l'étude comme l'évaluation de la perception

entourant la migration vers les LPL, comme nous l'avons fait, reflète l'accent mis dans le livre [WHH03] sur l'établissement d'objectifs de recherche clairs. La phase de *collecte de données*, impliquant la conception soignée d'entretiens centrés sur des sujets spécifiques, doit être mise en perspective avec les discussions du livre [WHH03] sur la sélection de méthodes appropriées de collecte de données adaptées au contexte de la recherche. En particulier, nous avons fait de notre mieux pour appliquer les recommandations du livre [WHH03] sur la planification rigoureuse de la collecte de données pour obtenir des résultats valides et fiables. L'*interprétation des données* fait référence à l'importance d'une analyse impartiale. S'engager dans des discussions approfondies, écouter attentivement et transcrire avec précision pour extraire des données qualitatives éclairantes correspond aux conseils du livre [WHH03] sur la réduction des biais d'interprétation.

La section méthodologie est composée de la présentation des participants (Sect. 3.3.1), de la conception des entretiens (Sect. 3.3.2), de notre approche pour mener les entretiens (Sect. 3.3.3), et de la méthode d'analyse des entretiens (Sect. 3.3.4).

3.3.1 Participants

Nous avons sélectionné les participants des entretiens afin d'avoir une population représentative [KP08]. Ils couvrent différents types d'acteurs du projet, différents niveaux d'expérience dans le projet et différentes formations académiques. Quatre types d'acteurs participent au projet : les responsables de produits (PO), les développeurs (IT), les agronomes (AGRO) et les concepteurs d'interface et d'expérience utilisateur (UX). Les acteurs ont des formations académiques différentes. Nous distinguons ici les personnes titulaires d'un diplôme d'ingénieur ou d'un master de celles titulaires d'un doctorat. Nous avons également sélectionné des participants ayant différents niveaux d'expérience dans le projet. Enfin, nous avons sélectionné 16 participants représentatifs, parmi lesquels neuf sont des informaticiens travaillant actuellement ou ayant travaillé par le passé sur le projet, deux sont des agronomes, trois sont des PO et deux sont des concepteurs d'interface et d'expérience utilisateur, tous travaillant sur le projet. Le tableau 3.1 montre la répartition des formations académiques des participants par rôle et présente les niveaux d'expérience dans l'entreprise.

La motivation des répondants est importante pour obtenir des réponses pertinentes, comme le soulignent Kitchenham et al [KP08]. Dans cette étude, les répondants savaient qu'ils recevraient un retour d'information et que leurs réponses seraient utilisées pour définir une bonne stratégie lors de la mise en œuvre du processus de migration.

3.3.2 Conception des entretiens

Nous avons opté pour des entretiens structurés avec une liste de questions. Nous avons fait de notre mieux pour suivre les principales recommandations en utilisant des termes compréhensibles et non ambigus [KP08]. Chaque question est liée à un concept

TABLE 3.1 – Rôles (colonne 1), formation académique (colonnes 2 et 3) et nombre moyen d'années de service (colonne 4) des participants.

Rôle	PhD	Ingénieur	nombre moyen # d'années de service (écart-type)
IT	2	7	6.1 (1.5)
PO	2	1	5.7 (2.6)
UX	0	2	8.5 (2.1)
AGRO	2	0	6 (1.4)
TOTAL	6	10	6.6 (2.2)

spécifique comme sur la réutilisabilité ou les pratiques par exemple. Nous avons évité les questions personnelles ou les interrogations sur des événements anciens. Nous avons également évité les questions inappropriées lorsque le répondant n'avait pas suffisamment de connaissances pour y répondre. Cet aspect nous a amené à considérer une partie des entretiens comme étant pilotée par les acteurs eux-mêmes. Le questionnaire des entretiens se compose de trois sections. La première section est une introduction où la personne interviewée se présente. La section principale suit, elle est divisée en deux sous-sections, l'une se concentrant sur le processus actuel et l'autre sur la migration. Enfin, il y a une section de conclusion qui couvre la discussion sur l'entretien dans son ensemble.

Entretiens dirigés par les rôles

Les acteurs du projet n'ont pas tous la même vision ni les mêmes compétences en génie logiciel. Pour prendre cela en compte, nous avons utilisé différentes variantes du questionnaire. Nous savions que les développeurs et les UX designers sont les profils les plus orientés génie logiciel, que les PO sont plus proches du domaine métier, et que les agronomes ont un profil technique en agronomie et en programmation des simulateurs, mais assez éloigné des préoccupations de réutilisation de code. Les agronomes ont été impliqués afin d'apporter un point de vue différent sur le projet de migration. Comme spécifié dans le tableau 3.2, le questionnaire d'entretien comprend à la fois des questions communes et spécifiques. Cela implique que certaines parties de l'entretien sont exclusivement destinées à des acteurs spécifiques.

Les questions communes sont des questions du début de l'entretien, sur la personne interrogée, sur son parcours au sein de l'entreprise, ses études et son expérience puis à la fin de l'entretien à propos de son ressenti sur la discussion. Les questions spécifiques sont des variantes correspondant à chaque rôle :

— Pour les développeurs : les questions se concentrent sur des qualités internes du code source, telles que la réutilisabilité ou la redondance.

TABLE 3.2 – Nombre de questions communes à tous les rôles (première ligne) et questions supplémentaires pour les rôles spécifiques (lignes suivantes).

Rôles des participants	Nombre de questions
Tous les rôles (questions communes)	12
IT	12
PO	8
UX	12
AGRO	12
TOTAL	56

— Pour les PO : les questions se concentrent sur la gestion de projet et les pratiques d'équipe dont les principes d'agilité.

— Pour les UI/UX designers : les questions se concentrent sur le développement de l'interface utilisateur.

— Pour les agronomes : les questions se concentrent sur le développement des simulateurs de prédiction et sur leur collaboration avec les développeurs.

Le nombre de questions conçues est détaillé dans le tableau 3.2.

Entretiens dirigés par les thèmes

Nous avons construit nos entretiens autour de quatre axes principaux liés à la migration afin d'identifier : (1) les bonnes pratiques actuelles, (2) les avantages attendus, (3) les risques perçus et (4) les craintes.

(1) Nous abordons les *bonnes pratiques actuelles* en posant les questions suivantes :

— Que pensez-vous des pratiques existantes ?

— Quels sont les points à conserver selon vous ?

— En termes de développement pour et par la réutilisation, quelle est votre opinion générale sur la base de code actuelle et sur la base de code commune elle-même ?

Dans la dernière question, l'énoncé « En ce qui concerne le développement pour et par la réutilisation » fait référence à la manière dont le développement de logiciels a été réalisé sur la plateforme. Cela signifie que les développeurs peuvent mettre l'accent sur la création de code dans le but de le rendre facilement réutilisable dans différents contextes (développement pour la réutilisation) ou bien préciser comment maximiser la construction de code en réutilisant et en assemblant du code existant (développement par la réutilisation).

(2) Les *avantages* concernant la nouvelle approche (suite à la migration) sont déduits des questions suivantes :

- Que pensez-vous d'une plateforme permettant de développer (dériver) rapidement des logiciels similaires ?
- Voyez-vous l'intérêt d'une telle approche dans les processus actuels de l'entreprise ?
- Quelles bonnes pratiques peuvent découler de cette approche ?

Poser ces questions concernant les avantages mettra potentiellement en évidence les changements attendus qui élimineront éventuellement les mauvaises pratiques dans le processus de développement actuel (avant la migration).

(3) Les *risques* perçus liés à l'approche envisagée sont déduits de la question suivante :

- Quels risques peuvent découler de cette approche sur vos modes de travail actuels ?

(4) Les *craintes* limitant l'enthousiasme sont déduites de la question suivante :

- Avez-vous des craintes liées à ce type de changement ?

3.3.3 Conduite des entretiens

Nous souhaitions que les entretiens se déroulent de manière fluide, davantage comme une discussion que comme de simples séances de questions-réponses. Nous avons planifié la conduite des entretiens en plusieurs parties.

La première partie des entretiens, avant de poser la première question, consiste en un bref rappel des principes de l'ingénierie des LPL et de l'objectif du questionnaire. Les premières questions portent sur le rôle de l'acteur interrogé au sein de l'entreprise. Le but avant tout est de comprendre le profil de l'acteur. Nous continuons par des questions générales sur le processus de développement au sein de l'entreprise. La partie suivante de l'entretien contient des questions plus précises sur le processus de développement actuel des nouveaux produits. Cette partie vise à comprendre la perception des forces actuelles, des bonnes pratiques et de l'état du processus. La discussion sur l'état actuel des pratiques constitue une transition parfaite pour aborder une nouvelle approche qui a pour but de de surmonter les faiblesses actuelles. La dernière partie est consacrée à une discussion sur la migration. Les participants sont encouragés à exprimer leurs craintes, les risques perçus et les avantages possibles. Les deux dernières questions visent à améliorer les entretiens et à conclure.

Tous les entretiens ont été menés par la même personne, moi-même. Chaque entretien a duré entre 30 minutes et 1 heure. Ils ont été appréciés et les personnes interrogées ont déclaré qu'elles se sentaient incluses dans le projet de migration grâce à la manière dont les entretiens étaient menés. Chaque entretien s'est conclu par la question : « Y

avait-il une question à laquelle vous vous attendiez et que je n'ai pas posée ? ». Cette question permet à la personne interrogée de réfléchir à toutes les questions et de participer à l'amélioration de la fluidité des entretiens suivants. Les questions restent inchangées pour garantir l'homogénéité de l'analyse des données. L'intention est de permettre aux participants de réfléchir à tous les aspects de l'entretien, tout en participant activement à l'amélioration des entretiens.

La transcription a été réalisée par prise de notes. Toujours en suivant les recommandations de [KP08], la personne qui faisant les entretiens, en l'occurrence, moi-même, a fait de son mieux pour veiller à ce que les questions et les échanges ne contiennent pas de biais, tels que l'influence du libellé des questions sur la personne interrogée, ou l'ordre des questions. Elle a également veillé à poser suffisamment de questions et à ne pas influencer la personne interrogée. Dans le cadre d'une amélioration continue, un premier entretien pilote a été réalisé avec notre principal collaborateur au sein de l'entreprise qui supervise ce travail de recherche avec nous, cela nous a permis d'améliorer le questionnaire et d'assurer la fluidité du processus. Cependant, les données de l'entretien pilote n'ont pas été prises en compte dans l'analyse.

3.3.4 Analyse des entretiens

Nous avons tout d'abord réalisé une analyse qualitative des résultats en lisant attentivement plusieurs fois les résumés des entretiens. Pendant les entretiens, nous avons enregistré les discussions, pris des notes manuscrites, puis résumé le contenu enregistré. Cette approche combinée vise à garantir que toutes les informations pertinentes soient capturées et à minimiser le risque de ne pas tenir compte des détails critiques, ce qui est conforme à la pratique courante et recommandée d'enregistrer et de transcrire les entretiens dans la recherche qualitative. Le processus d'analyse a été effectué par l'équipe de recherche, qui comprenait la personne qui a mené les entretiens.

Ensuite, nous avons conçu un ensemble de mesures pour effectuer une analyse quantitative. Ces mesures sont une combinaison de *comptage des réponses* et de *fréquence*. Tout d'abord, nous avons compté *combien de bonnes pratiques actuelles sont identifiées*, puis *combien d'avantages sont identifiés ?*, ensuite *combien de risques sont identifiés ?* et enfin *combien de craintes sont identifiées ?*. Pour répondre à ces questions, nous avons recherché des formulations similaires de réponses et nous avons interprété les réponses des entretiens pour extraire des mots clés simples résumant les idées. Par exemple, *« La généricité pousse à factoriser de plus en plus »* est résumé par *« généricité »* et *« factorisation »*. Pour chaque thème, nous avons d'abord identifié la fréquence des termes génériques exprimés. Ensuite, nous les avons séparés par rôle, c'est-à-dire selon l'acteur qui les a exprimés.

Les mots utilisés pour résumer les réponses ont été établis manuellement, selon nos connaissances sur les méthodes agiles et l'ingénierie des LPL. Ces mots ont été validés

a posteriori, en les identifiant dans un ensemble de documents de référence [2], qui sont : *Le manifeste Agile* [BBvB+01], *Software engineering - principles and practice* [vV08] et *Software Product Line Engineering - Foundations, Principles, and Techniques* [PBvdL05].

Les mots clefs ont été identifiés avec une méthode composée des étapes suivantes :

1. Suppression des mots vides et de la ponctuation
2. Tri des mots par fréquence
3. Sélection des mots les plus significatifs
4. Parcours les documents pour valider le contexte
5. Enrichir pour former des mots composés
6. Vérification de la cohérence

Les mots clés potentiels ont été identifiés avec une méthode composée de plusieurs étapes. Nous avons d'abord trié les mots par fréquence *(2)* après avoir supprimé les mots vides et la ponctuation *(2)*. Nous avons sélectionné les mots les plus significatifs par rapport à la question posée *(3)*, puis nous avons parcouru les documents pour valider le contexte d'utilisation de chaque mot *(4)*. Nos mots identifiés ont été enrichis pour former des mots composés en ajoutant des adverbes *(5)*, comme *« trop complexe »* au lieu de *« complexe »* pour renforcer la signification du mot. Nous avons comparé les mots déduits des livres avec les mots des réponses *(6)*. Tout d'abord, nous avons vérifié s'il y avait une correspondance entre eux. En cas de correspondance négative, nous avons cherché des mots plus adaptés. Par exemple, nous avons remplacé « inutilisable »par « inutile »pour mieux correspondre aux mots identifiés dans les livres. En cas de correspondance positive, nous avons conclu que le mot était correctement aligné avec les mots clés potentiels identifiés. Ce processus nous a permis d'affiner et d'améliorer les réponses synthétisées avec les connaissances issues des livres et d'obtenir une meilleure correspondance avec le langage utilisé dans ces livres. De plus, cette approche nous a aidés à extraire les termes les plus pertinents et significatifs pour améliorer notre analyse des données.

Pour réaliser l'analyse de fréquence et le traitement de similarité, nous avons développé un script en Python dans un *Notebook Jupyter*. Une fois les termes du livre triés par fréquence, nous avons choisi manuellement les mots significatifs.

Les listes de sujets sont disponibles avec nos données dans un répertoire en ligne : https://gite.lirmm.fr/tgeorges/interviewartefact.
Le répertoire comprend les questions et les réponses correspondantes, l'analyse des entretiens, les listes de sujets, le Notebook et plusieurs graphiques représentant les réponses.

2. Les mots liés aux sentiments et aux risques ont plutôt été identifiés simplement à partir d'un dictionnaire.

3.4 Résultats des entretiens

Dans cette section, nous analysons les résultats par question de recherche : les *bonnes pratiques* actuelles (Sect. 3.4.1), les *avantages* (Sect. 3.4.2), les *risques* (Sect. 3.4.3) et les *craintes* (Sect. 3.4.4) perçus pour la future LPL.

Le tableau 3.3 détaille le nombre de bonnes pratiques, d'avantages, de risques et de craintes identifiés qui sont ensuite analysés. Les résultats sont présentés en fonction des sujets et des rôles, suivant la méthodologie qui a guidé le processus de conception.

Les réponses fournies par les interviewés sont réparties de manière équilibrée. Lorsque nous analysons les réponses en fonction du nombre d'années d'expérience, les individus les moins et les plus expérimentés n'ont pas fourni une quantité ni une qualité significativement différente de réponses. Par exemple, lorsque nous avons demandé « Quelles bonnes pratiques peuvent être dérivées de cette approche ? », les développeurs ayant soit 4 ans d'expérience, soit 7 ans d'expérience ont fourni les mêmes réponses. Plusieurs diagrammes en barres représentant les réponses en fonction du nombre d'années d'expérience sont disponibles dans le répertoire partagé.

Table 3.3 – Synthèse des résultats : nombre identifié de bonnes pratiques actuelles, de bénéfices, de risques et de craintes, par acteur, puis agrégés.

	IT	PO	UX	Agro	total
Bonnes pratiques actuelles	20	6	5	3	34
Avantages	21	6	5	1	33
Risques	13	6	3	2	24
Craintes	8	6	0	1	15

3.4.1 Les bonnes pratiques actuelles (RQ1)

Cette partie des entretiens visait à détecter les bonnes pratiques actuelles qui sont essentielles, afin de les maintenir dans la nouvelle approche. La pratique la plus fréquemment citée est la **Factorisation et la Réutilisation**. Cela s'explique par des habitudes de développement au sein de l'entreprise, ITK. En effet, chaque nouvelle fonctionnalité est discutée au sein de l'équipe pour déterminer si elle est spécifique à une application ou plus générique et disponible pour toutes les applications. **L'innovation** est une idée qui motive les équipes à maintenir leurs connaissances à jour. C'est important pour les participants et ils ont la possibilité de mettre régulièrement à jour la base de code avec de nouvelles fonctionnalités et des mises à jour de frameworks. L'innovation concerne les aspects techniques des IT et les avancées en agronomie également. A ITK l'innovation englobe la recherche, le développement, la mise en œuvre de nouvelles idées, ainsi que l'adaptation de solutions existantes pour répondre aux besoins changeants du marché, des clients ou des utilisateurs finaux. Le développement **rapide** est l'idée de base

du projet. Il implique la capacité de produire une nouvelle application plus rapidement que la précédente. L'**esprit du génie logiciel** regroupe tous les principes détaillés ci-dessus, c'est-à-dire la factorisation, la réutilisation, l'innovation et la rapidité. Le projet a été initialement pensé et conçu comme une plateforme logicielle avec une architecture et des artefacts réutilisables communs. Cette notion implique l'utilisation des bonnes pratiques du génie logiciel. La proximité avec les objectifs de la LPL contribue à promouvoir le projet de migration. Nous avons identifié **7** bonnes pratiques actuelles différentes. Nous avons remarqué que la *Factorisation* et la *Réutilisation* sont les deux idées les plus fréquemment mentionnées avec respectivement 9 et 8 occurrences (tableau 3.4). La figure 3.1 met en évidence graphiquement la répartition des mentions.

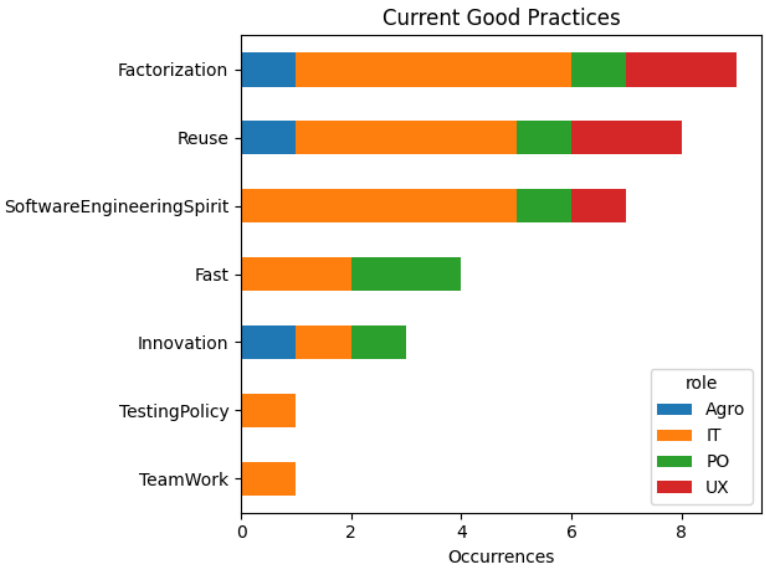


FIGURE 3.1 – Idées les plus fréquemment mentionnées à propos des bonnes pratiques.

3.4.2 Les bénéfices (RQ2)

Les avantages perçus sont importants à identifier et à prendre en compte car leur expression sera motivante pour toute l'équipe. La **personnalisation** est l'une des principales caractéristiques d'une LPL et a été l'idée la plus fréquemment mentionnée par les *ITs*, les *UXs* et les *AGROs*. Une LPL est perçue comme une plateforme capable de générer un logiciel sur mesure à partir d'une collection de logiciels existants. La **rapidité**, mentionnée par les *ITs*, les *POs* et les *UXs*, est la deuxième idée la plus fréquemment mentionnée. Elle correspond à la capacité de dériver rapidement un nouveau logiciel. **L'automatisation** a également été mentionnée. Il s'agit du processus de dérivation automatique de nouveaux logiciels. Les *ITs* et les *POs* ont lié l'automatisation à la *rapidité*.

TABLE 3.4 – Idées les plus fréquemment mentionnées à propos des bonnes pratiques actuelles, par acteur, puis agrégées.

Rôle	IT	PO	UX	AGRO	TOTAL
Factorisation	5	1	2	1	9
Réutilisation	5	1	2	0	8
Esprit génie logiciel	4	1	1	1	7
Rapidité	2	2	0	0	4
Innovation	1	1	0	1	3
Travail d'équipe	1	0	0	0	1
Politique de test	1	0	0	0	1

L'**industrialisation** a été mentionnée par les *ITs* et les *UXs*, signifiant ici la production à la demande et automatisée d'applications à faible coût. L'amélioration de la **qualité du code** a été mentionnée uniquement par les *ITs*. La qualité du code prend ici en compte l'homogénéité du code, la testabilité et la fiabilité. Les **avantages commerciaux (ou la rapidité de mise sur le marché)** ont été mentionnés par les *ITs* et les *POs* et regroupent toutes les caractéristiques permettant d'améliorer l'aspect commercial. Cela peut se référer, par exemple, à la vente de plus de logiciels ou à une réponse rapide à un appel d'offres. Ces **7** avantages sont mis en évidence dans la figure 3.2. Nous avons remarqué que la *personnalisation* est l'idée la plus fréquemment exprimée, mentionnée 10 fois, suivie de *Rapidité* (voir le Tableau 3.5).

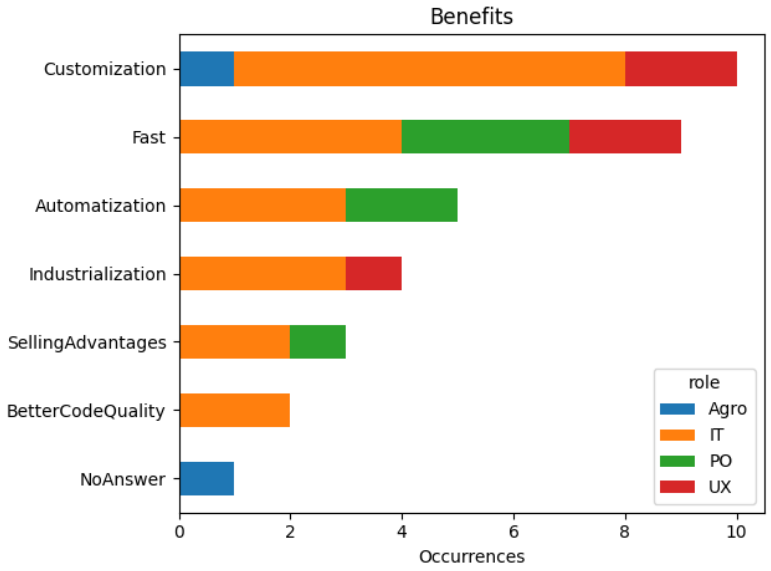


FIGURE 3.2 – Idées les plus fréquemment mentionnées à propos des bénéfices attendus.

TABLE 3.5 – Idées les plus fréquemment mentionnées à propos des bénéfices futurs, par acteurs, puis agrégés.

Rôle	IT	PO	UX	AGRO	TOTAL
Personnalisation	7	0	2	1	10
Rapidité	4	3	2	0	9
Automatisation	3	2	0	0	5
Industrialisation	3	0	1	0	4
Meilleure qualité de code	2	0	0	0	2
Avantages commerciaux	2	1	0	0	3
Pas de réponse	0	0	0	1	1

3.4.3 les risques (RQ3)

Les risques sont les préoccupations, les faiblesses potentielles apportées par la migration et il est important de les identifier pour pouvoir les éviter ou du moins les limiter. Le risque le plus fréquemment mentionné est la **complexité**, citée par les *ITs* et les *POs*. Il n'est pas intuitif de comprendre comment fonctionne une LPL et quels sont tous les processus impliqués. La deuxième idée la plus fréquemment exprimée est **la difficulté de maintenir et de faire évoluer** la LPL en la voyant comme une boîte noire, ce qui limite la possibilité de faire des évaluations. Cela a été mentionné par les *ITs* et les *UXs*. Ensuite, la **perte de fonctionnalités**, exprimée par les *ITs* et les *AGROs*, correspond à l'incapacité de récupérer toutes les fonctionnalités actuelles et même de recréer les applications actuelles existantes. La **perte de créativité** a été exprimée par les *UXs*. Elle résulte du manque de confiance dans la capacité de la LPL à évoluer et du fait qu'elle pourrait limiter les types d'évolution possibles. Mentionné uniquement par les *POs*, l'**excès d'effets de bords** est dû à la factorisation des fonctionnalités. Une modification de fonctionnalité peut avoir un impact sur toutes les applications qui l'utilisent et peut propager des bugs. Enfin, l'aspect **chronophage** de la migration vers une LPL a été mentionné par les *POs* qui s'inquiétaient de l'équilibre entre les avantages et le temps nécessaire pour mettre en œuvre la LPL. La figure 3.3 montre les **7** risques identifiés. Nous avons remarqué que la **complexité** était le risque le plus fréquemment mentionné, suivi de la **difficulté à maintenir** (tableau 3.6).

3.4.4 Les craintes (RQ4)

Les craintes sont des sentiments négatifs personnels et **subjectifs** liés à la migration, qu'il est important de prendre en compte pour rassurer les acteurs. Lors de la réponse aux deux premières questions concernant les avantages, les personnes interrogées ont

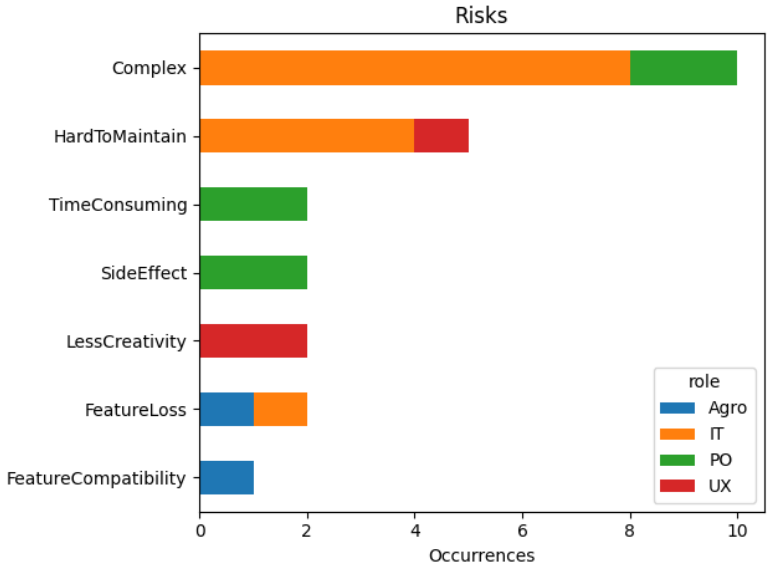


FIGURE 3.3 – Idées les plus fréquemment mentionnées à propos des risques.

souligné les craintes induites par les risques qu'ils ont identifiés. Deux questions clés qui ont indirectement contribué à l'identification des craintes sont les suivantes :

— Voyez-vous l'intérêt d'une telle approche dans les processus actuels de l'entreprise ?

— Que pensez-vous d'une plateforme permettant de développer (dériver) rapidement des logiciels similaires ?

Aucun des *UXs* n'a exprimé de craintes contrairement aux *POs*, qui ont tous exprimé des inquiétudes. La crainte la plus fréquemment exprimée est la création d'un système **inutile** (la LPL elle-même), trop éloigné de la réalité et empêchant toute évolution. Une

TABLE 3.6 – Idées les plus fréquemment mentionnées à propos des risques, par acteur, puis agrégées.

Rôle	IT	PO	UX	AGRO	TOTAL
Complexité	8	2	0	0	9
Difficile à maintenir	4	0	1	0	5
Perte de fonctionnalités	1	0	0	2	3
Moins de créativité	0	0	2	0	2
Incompatibilité de fonctionnalités	0	0	0	1	1
Effets de bord	0	2	0	0	2
Chronophage	0	2	0	0	2

autre crainte est l'**excès de complexité ajoutée** du projet, limitant l'intérêt et l'implication des acteurs. Ils signalent également la **résistance au changement**, une crainte commune à tous les nouveaux projets. Les acteurs sont également préoccupés par le fait que le projet puisse être plus **coûteux** que rentable. Enfin, ils s'inquiètent du fait que le résultat soit *moins complet* que le processus actuel.

Les **7** différentes craintes identifiées sont présentées dans la figure 3.4. Nous avons remarqué que les principales craintes des ITs sont **Inutile**, **Excès de complexité** et **Résistance au changement** (voir tableau 3.7).

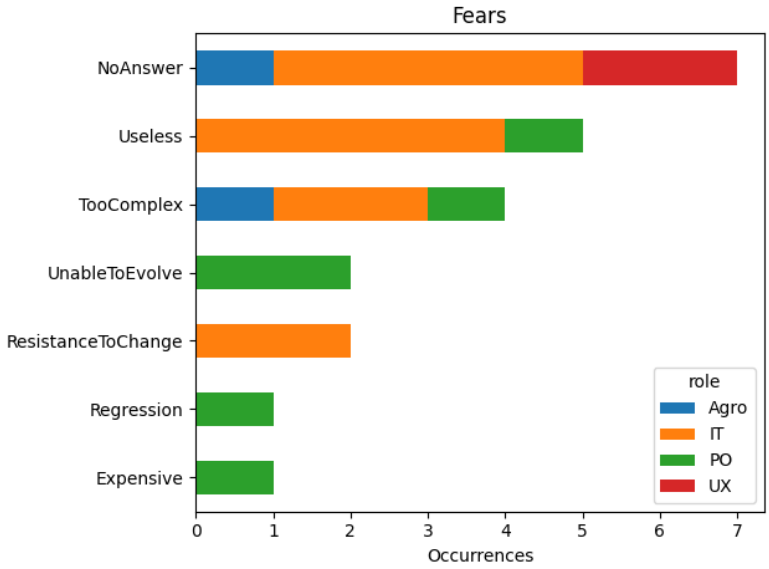


FIGURE 3.4 – Idées les plus fréquemment mentionnées à propos des peurs.

TABLE 3.7 – Idées les plus fréquemment mentionnées à propos des craintes, par acteur, puis agrégées.

Rôle	IT	PO	UX	AGRO	TOTAL
Pas de réponse	4	0	2	1	7
Inutile	4	1	0	0	5
Excès de complexité	2	1	0	1	4
Résitance au changement	2	0	0	0	2
Coûteux	0	1	0	0	1
Incapcité à faire évoluer	0	2	0	0	2
Régression	0	1	0	0	1

Finalement, les acteurs, quel que soit leur rôle, ont exprimé le besoin de formation

et de preuve de concepts. Ils souhaitent mieux évaluer l'utilité d'une LPL dans leurs processus afin de comprendre et d'adopter la nouvelle approche.

3.5 Directions pour préparer la migration vers une LPL

Sur la base des bonnes pratiques, des avantages, des risques et des craintes identifiés, nous proposons ici des lignes directrices pour préparer et mener au mieux la migration vers une LPL. Ces lignes directrices ont été dérivées des discussions et des décisions prises au sein de l'entreprise.

Dans cette section, nous présentons les résultats attendus (Sect. 3.5.1), suivis de la manière d'exploiter les bonnes pratiques actuelles (Sect. 3.5.2), d'encourager les bénéfices futurs possibles (Sect. 3.5.3), de réduire les risques (Sect. 3.5.4), d'atténuer les craintes (Sect. 3.5.5), puis nous introduisons des actions concrètes à mener (Sect. 3.5.6) et enfin nous terminons par une discussion (Sect. 3.5.7).

3.5.1 Résultats attendus

Avant de commencer les entretiens, nous nous sommes demandés quels types de résultats pouvaient être attendus. Nous avons pensé que les personnes ayant des connaissances ou une affinité avec la réutilisabilité seraient les plus positives et intéressées. Elles devraient être motivées pour initier et encourager la mise en œuvre du projet. Dans notre cas, les développeurs ayant le plus haut niveau d'expérience ont fourni davantage de réponses liées à la mentalité, tandis que les développeurs ayant le moins d'expérience se sont davantage concentrés sur les aspects techniques. Cependant, cette variation liée à l'expérience n'est pas significative, tant en termes de quantité que de qualité des réponses fournies.

Les personnes proches du domaine métier ou moins techniques devraient être les plus réticentes, car même en admettant l'utilité, elles pourraient avoir davantage de préoccupations concernant la rapidité de livraison. Étant donné que l'équipe est relativement petite et habituée à travailler sur de nouvelles technologies, et étant donné que l'utilisation d'une LPL a été envisagée dès le début du projet, nous nous attendions à une faible résistance au changement par rapport à une migration habituelle. Dans ce contexte, et en utilisant les résultats des entretiens, nous avons identifié des pistes (directions) pour consolider les bonnes pratiques, accroître les avantages et limiter les risques et les craintes.

3.5.2 Exploitation des bonnes pratiques actuelles

Mettre en avant et encourager les bonnes pratiques existantes lors de la mise en œuvre de la LPL et de sa maintenance est un facteur rassurant. Les habitudes évolueront

mais une équipe qui ne change pas ses bonnes pratiques de développement est plus susceptible d'être motivée et d'accepter la nouveauté. Les habitudes de l'équipe connaîtront des changements significatifs, et en maintenant les bonnes pratiques actuelles, l'objectif est de maintenir l'équipe dans un environnement familier.

La factorisation, la réutilisation et la rapidité sont les trois fondements des LPL, comme indiqué dans [PBvdL05]. Dans notre contexte, ces éléments sont déjà des bonnes pratiques établies, nous pouvons donc les utiliser pour mettre en avant les avantages futurs. La migration vers une LPL est un bon moyen de rendre la réutilisation et la factorisation durables.

L'innovation fait déjà partie intégrante du processus actuel. L'efficacité d'une LPL devrait encourager son adoption. La motivation peut venir du fait que le temps économisé en abandonnant l'approche chronophage du *copier et s'approprier* (*clone-and-own* [DRB+13], en anglais) peut être utilisé pour le développement de fonctionnalités plus innovantes. De plus, développer dans le contexte d'une LPL peut renforcer la cohésion de l'équipe autour d'un paradigme novateur.

3.5.3 Encourager les futurs bénéfices potentiels

Parmi les avantages évoqués, nous avons identifié d'autres bénéfices des LPL tels que la *personnalisation*, l'*automatisation* et l'*industrialisation* [PBvdL05]. Il est également possible de discuter et d'inclure d'autres critères. Par exemple, une *meilleure qualité du code* peut être encouragée en utilisant l'avantage de résorber la dette technique. Une base de code plus homogène, cohérente et avec moins de bugs est obtenue en évitant les situations de *clone-and-own*. La correction des bugs sur la plateforme commune améliore également la qualité du code de manière globale.

Les avantages en termes de vente sont étonnamment aussi mentionnés par les *POs*, même s'ils sont davantage mentionnés par les *ITs*. Par exemple, l'utilisation de ce type d'ingénierie peut confirmer et consolider l'image de recherche et développement (R&D) de l'entreprise. Un autre exemple est que l'adoption d'une LPL favorise une production plus rapide de nouveaux logiciels. De plus, elle encourage la création de nouveaux produits avec des innovations pour chaque marché d'activité similaire. Ces deux exemples peuvent motiver les équipes à adopter une LPL.

3.5.4 Réduire les risques

Réduire les risques est essentiel pour limiter la résistance au changement. Voici quelques pistes pour les réduire. Toutes les craintes, qu'elles soient liées ou non à ces risques, doivent être prises en compte, car elles empêchent une bonne adoption.

Certains risques ou faiblesses sont récurrents lors de l'adoption d'une LPL. Cependant, ils ne sont pas insurmontables si nous suivons et guidons la migration en les prenant en considération.

Le sentiment de *complexité* s'explique par une mauvaise compréhension de la LPL. Cependant, une LPL peut poser des défis complexes pour ceux qui sont familiers avec l'ingénierie des LPL. La complexité peut exister au sein de la LPL elle-même, mais elle ne devrait pas dépasser la complexité de la base de code sur laquelle le personnel informatique travaille déjà et qu'ils maîtrisent collectivement. Au sein de la LPL, certaines hypothèses concernant la variabilité sont rendues explicites à travers des modèles, visant à améliorer la « compréhensibilité »et par conséquent à traiter la complexité.

Nous pensons que la demande de formation et de preuve de concept peut être liée à ce sentiment. Nous expliquons donc qu'au contraire, une LPL permet de gagner du temps pour mettre en œuvre de nouvelles fonctionnalités et réaliser des développements plus créatifs. En détaillant cela, nous évitons les risques de *perte de temps créatif* tels qu'exprimés par les *UXs* et de *perte de temps pour le développement innovant*. Les *effets de bords* sont évitables grâce à des tests de non-régression systématiques.

3.5.5 Atténuer les craintes

Les actions concrètes facilitent une transition plus fluide et sont largement utilisées dans divers cycles de vie du développement de logiciels, en particulier lors de la migration vers une LPL.

Les craintes peuvent être évitées en prenant le temps d'écouter les inquiétudes de l'équipe et de discuter des points faibles et des points forts [Wei82]. De plus, elles sont perçues comme des biais exprimés par les personnes interrogées.

Considérer une LPL comme un système *trop complexe à utiliser et à faire évoluer, rendant la nouvelle approche inutile* peut être compris. La confiance peut être rétablie en donnant accès à la LPL en tant que boîte blanche, avec de nombreuses explications et une documentation riche, tout en incluant tous les acteurs possibles dans le projet. L'inclusion des acteurs commence par leur demander de participer à la migration. Ensuite, il est important de les encourager à faire des compte-rendus réguliers à chaque étape du processus. Cela leur permettra de sentir qu'ils sont écoutés et compris, et de leur apporter une assistance en cas de besoin.

3.5.6 Actions concrètes

De nombreuses pratiques agiles peuvent être utilisées pour minimiser les effets de l'adoption de la migration, par exemple, un Sprint Zero [Boe10] est une possibilité pour améliorer le travail d'équipe pendant la migration. Ce sprint comprend différentes étapes. La première étape consiste à établir un *accord de travail* où nous planifions les rituels agiles pour aider l'équipe à réaliser la migration. Ensuite, avoir une *vision* claire et commune est une condition préalable pour s'assurer que tous les membres de l'équipe ont compris l'objectif et les avantages de la LPL. La troisième étape consiste à créer les *personas*, c'est-à-dire les profils-types des utilisateurs de la LPL, qui sont déterminés avant

le début du projet pour favoriser l'empathie et savoir qui l'utilisera. L'étape suivante consiste à construire le *backlog* (liste des tâches à accomplir) en tenant compte de tout le travail nécessaire. Ensuite, chaque tâche du *backlog* doit être *estimée* et intégrée dans une feuille de route. Planifier des rétrospectives récurrentes pour que l'équipe puisse partager ses sentiments est un moyen d'améliorer le travail d'équipe.

Certaines pratiques de génie logiciel sont utiles pour améliorer la qualité du projet, comme des sessions régulières de *refactoring* pour augmenter les pratiques de réutilisation et éliminer la dette technique. Ce sont des pratiques visant à garantir un code de meilleure qualité : planifier régulièrement des sessions de tests pour éviter les bugs, prévoir des sessions de tests d'acceptation avec les utilisateurs finaux est également un bon moyen de s'assurer que nous sommes sur la même longueur d'onde. Surveiller constamment l'exécution du code est une possibilité de maintenir l'application aussi efficace et évolutive que possible.

Organiser des sessions de test d'outils peut donner aux acteurs l'occasion de se familiariser avec les outils et de comprendre leurs capacités [Cha96]. Dans le cas de l'adoption d'une LPL : le framework clone-and-own Ecco [FLLE15], Feature IDE [TKB+14] pour construire des modèles de caractéristiques et des variantes, Mobioos Forge [GZ22] pour implémenter une LPL ou le framework But4reuse [MZB+17] pour la rétro-ingénierie des LPL.

Ces actions concrètes sont bien sûr générales et peuvent être appliquées à différents contextes de développement de logiciels, qu'ils soient basés sur une LPL ou non. Nous voyons des avantages renforcés dans un environnement de développement basé sur une LPL, car l'amélioration de qualité est intégrée au cœur de la base de code. Cela profite aux applications actuelles, mais bénéficiera également aux applications dérivées futures.

3.5.7 Discussion

Le chemin vers la migration vers une LPL s'étend sur une période significative. Il débute par une phase initiale marquée par les pratiques et habitudes prédominantes, suivie de discussions pour évaluer la viabilité de la migration et si elle constitue une décision prudente. La phase de migration qui suit est celle que nous étudions. Au cours de celle-ci nous avons travaillé à recueillir le ressenti et à surmonter la résistance au changement pendant la migration du code source. La dernière étape est la phase post-migration qui marque le début d'un nouveau processus de développement.

La transition vers une approche LPL engendre d'importantes modifications organisationnelles. Avec l'adoption d'une LPL, les équipes deviennent plus transversales, collaborant sur un ensemble partagé de composants de base réutilisables à travers plusieurs projets. Ce changement permet une utilisation efficace des ressources, un raccourcissement du délai de mise sur le marché et une focalisation sur l'innovation. L'accent mis sur la réutilisation des composants, la personnalisation basée sur la configuration et la documentation unifiée rationalisent les processus et renforcent l'efficacité. Bien que cette

transition nécessite une planification minutieuse et une gestion du changement, elle favorise finalement une culture de collaboration, d'efficacité et d'adaptabilité au sein de l'organisation.

Dans le contexte d'ITK, les résultats et les lignes directrices ont été présentés et discutés avec l'équipe de développement. Ils ont exprimé le souhait d'être bien informés et impliqués, et, en réponse, nous avons entretenu une collaboration continue et une progression en parallèle avec eux tout au long du processus de support à la migration. Nous pouvons mettre les résultats et les réflexions faites chez ITK en perspective avec plusieurs rapports d'adoption de LPL, de stratégies et d'expériences de la littérature.

Par exemple, Böckle et al. [BMK+02, BPL05] ont proposé une stratégie en quatre phases : Identification des parties prenantes, Motivation pour l'adoption de la ligne de produits, Élaboration d'un plan d'adoption et Lancement de la LPL. Le *Framework for Software Product Line Practice* introduit des pratiques pour mettre en œuvre avec succès un changement technologique sous différents aspects : vision, compétences, incitations, ressources et plans d'action. Nous avons déjà identifié les parties prenantes, en particulier celles qui jouent un rôle actif, comme les développeurs ou les Product Owners. Au début du projet, des discussions ont eu lieu avec les parties prenantes du personnel financier et de la direction. Dans cette étude, nous sommes actuellement à l'étape équivalente à la *Motivation pour l'adoption de la ligne de produits* et nous sommes en train de formuler le *plan d'adoption.*

Le plan d'action d'ITK adhère au *plan d'amélioration IDEAL* [McF96]. Le projet a commencé par la décision collégiale de *démarrer* la migration. Notre travail commence dès lors en *identifiant* les pratiques et la perception du projet. Les prochaines étapes de l'adoption par ITK consistent à *établir* une stratégie et un plan d'action, à *mettre en œuvre* le plan, et enfin, de manière classique dans une approche agile, à *apprendre* des succès et des échecs pour améliorer continuellement le processus de développement. Nos directives servent de méthode pour orienter la conception de l'établissement du plan d'action.

Il existe plusieurs différences entre nos obstacles et ceux identifiés précédemment dans [NJ13]. Par exemple, le *manque de talent nécessaire* ou le *manque de soutien managérial suffisant* sont absents de nos perceptions détectées en raison des spécificités de l'entreprise. Dans [NJ13], une stratégie d'entretien préconisée implique d'être conscient de divers aspects tels que « Contexte », « Avantages », « Défis », « Risques », « Stratégies »et « Transformations technologiques ». Dans la conception de nos entretiens, nous partageons avec eux certains éléments comme les « avantages »et les « risques », tandis que d'autres, comme les « obstacles », sont proches de nos « craintes ».

3.6 Étude de la validité

Dans cette section, nous rapportons les biais potentiels que nous avons identifiés dans notre méthode de recherche. Nous discutons également de la manière dont nous les avons pris en compte afin de les atténuer et de fournir une étude empirique objective, conformément aux recommandations de [WRH+12]. Nous examinons en particulier les menaces à la validité de la construction (Section 3.6.1), la validité interne (Section 3.6.2), la validité externe (Section 3.6.3) et la validité des conclusions (Section 3.6.4).

3.6.1 Validité de construction

Nous examinons ici si les questionnaires et les métriques capturent les concepts qu'ils sont censés évaluer.

Nous avons expliqué dans la section 3.3 comment les questions correspondent aux quatre axes des questions de recherche. L'ingénierie des LPL était inconnue de la plupart des personnes interrogées. Nous avons donc décrit les concepts et les objectifs de l'ingénierie des LPL pendant les entretiens. Cela a introduit un biais potentiel lors de l'explication. Cependant, nous faisons confiance aux personnes interrogées pour être capables de se forger leur propre opinion sur le sujet, et à la personne qui fait passer les entretiens pour éviter d'avoir trop d'impact sur cette opinion.

Les questions sont délibérément ouvertes afin d'encourager des réponses approfondies et de capturer un large éventail de concepts. Elles ont été soigneusement élaborées pour éviter de limiter l'étendue des réponses potentielles.

Les métriques ont été construites pour tirer le meilleur parti des informations que nous avons obtenues, et pour extraire autant de données utiles que possible. Les métriques comptent les termes pertinents et leur nombre d'occurrences (agrégé et par rôle) dans les réponses. Ainsi, les tableaux contiennent des informations exhaustives et les graphiques à barres mettent en évidence la distribution des valeurs.

Même si une réponse est unique et n'est partagée par personne d'autre, elle est tout de même considérée et prise en compte. Nous menons une réflexion exhaustive sur l'ensemble des réponses fournies.

Les termes extraits des entretiens ont été regroupés et alignés avec des termes du domaine de l'ingénierie des LPL, tels que *« personnalisation »* et *« rendre spécifiques certaines parties du code »*, qui correspondent à « *customization* ». Cela a permis d'uniformiser les résultats. De plus, cet alignement a aidé à trouver dans la littérature les arguments pour rassurer les acteurs relativement à leurs préoccupations [LKHB21, MZB+15, Kru92].

3.6.2 Validité interne

Nous rapportons ici les possibles biais dans la collecte et l'analyse des données. Les entretiens ont été organisés avec un petit panel de 15% du personnel de la PME, soit

50% du personnel informatique, 30% des PO, 10% des agronomes et tous les designers UI/UX. Le personnel travaillant sur le projet change au fil du temps, principalement en passant d'un projet à l'autre au sein de l'entreprise, les équipes évoluent. Nous avons discuté avec des personnes présentes depuis le début du développement de la base de code, mais aussi avec des personnes arrivées ou parties en cours de route. Le panel des personnes interrogées est représentatif des rôles et des différentes périodes du projet. Les résultats de ces discussions ont ensuite été interprétés et analysés par la personne qui fait passer les entretiens, mais ils ont également été discutés avec les membre de l'équipe de recherche afin de limiter les malentendus.

Le fait d'itérer ce processus avec de nombreuses personnes différentes nous a permis d'améliorer positivement la fluidité des entretiens. Les questions n'ont pas été modifiées, mais la manière dont les questions ont été abordées s'est améliorée avec le temps. Par exemple, cela implique de proposer des indices supplémentaires et d'approfondir les détails lorsque la personne interrogée n'est pas sûre d'elle-même, ou d'améliorer les transitions entre les questions. Malgré cela, nous pensons que la quantité et la qualité des informations extraites lors de ces entretiens étaient les mêmes, que ce soit lors du premier ou du dernier questionnaire.

Les craintes exprimées par les personnes interrogées peuvent être perçues comme un biais dans les résultats. Elles sont des sentiments subjectifs et dépendent largement de la personnalité et de l'expérience des acteurs du projet. Cependant, étant donné que nous avons interrogé 16 personnes et présenté des réponses agrégées, le biais est légèrement atténué.

3.6.3 Validité externe

Dans la validité externe, nous cherchons à évaluer ce qui peut être généralisé dans ce travail dans d'autres contextes de migration de LPL.

Nos questionnaires contiennent des questions spécifiques à l'entreprise. Elles sont liées au processus de développement pratiqué par l'entreprise. Mis à part cela, les autres questions sont liées à la réutilisation de code et aux LPL en général. Nous pensons que le processus de développement de l'entreprise, qui fait partie d'une méthodologie de gestion de projet plus large basée sur SCRUM [SB01], est courant dans de nombreuses entreprises de même taille ou plus petites. Nous pensons donc que les résultats discutés précédemment peuvent être généralisés à d'autres contextes d'entreprise utilisant des méthodologies agiles et organisant leurs équipes de développement de la même manière (IT, PO, ...).

Notre étude se concentre principalement sur une PME qui collabore avec notre équipe de recherche. Tous les participants sont de la même entreprise et sont interrogés sur le même projet de LPL. Néanmoins, nous avons conçu le processus et les lignes directrices de manière à ce qu'ils soient suffisamment larges pour être appliqués à d'autres cas de migration de LPL.

La réalisation d'entretiens personnalisés en fonction du rôle et des responsabilités des personnes interrogées, puis l'analyse unique et centralisée des résultats, est un bon moyen d'impliquer les personnes interrogées et d'obtenir des réponses plus précises. Cela a permis de collecter un ensemble de données de bonne qualité qui guide les discussions sur la manière de préparer et d'adapter la migration de LPL aux différents besoins. Toutes ces données sont disponibles en ligne pour permettre la reproduction de notre méthode dans d'autres contextes :
https://gite.lirmm.fr/tgeorges/interviewartefact.

3.6.4 Validité des conclusions

En ce qui concerne la validité des conclusions, nous évaluons dans quelle mesure nous tirons des conclusions raisonnables et déduisons des orientations pertinentes pour la migration et pour faire face à une éventuelle résistance.

Les bonnes pratiques, les avantages, les risques et les craintes identifiés peuvent être mis en perspective par rapport aux approches visant à surmonter la résistance au changement [KL04] : la stratégie Alpha consiste à augmenter la motivation et la stratégie Omega consiste à diminuer l'évitement. Les acteurs ont identifié des bonnes pratiques qui sont recommandées pour les projets agiles [BBvB+01], ce qui est en accord avec les habitudes de l'entreprise et avec la recommandation *Emphasize (mise en valeur) and Commitment (engagement)* de la stratégie Alpha. Les avantages identifiés sont des avantages traditionnels des LPL et leur encouragement soutient un autre conseil de la stratégie Alpha, qui est d'*ajouter des incitations*. De même, les risques et les craintes identifiés ont du sens. La demande des différents acteurs de disposer de solutions concrètes pour se rassurer face à ces risques et craintes, telles que la formation, les exemples et les démonstrations qui serviront de preuves de concept, correspond à *donner des garanties et renforcer l'estime de soi* de la stratégie Omega. Cela contribuera également à *améliorer la crédibilité de la source* de la stratégie Alpha.

3.7 Travaux connexes

Dans cette section, nous positionnons successivement notre travail dans les domaines de l'adoption des Lignes de Produit Logiciel, de la conception d'entretiens, de l'agilité et de l'ingénierie logicielle empirique.

Adoption des LPL Cette étape se situe entre l'étude de la pertinence de l'adoption des LPL, telle que celle discutée dans [RMS18, RMS19], et l'évaluation de la maturité ou de la post-migration, comme cela est rapporté dans [LKHB21]. Lindohf et al. [LKHB21] partagent des questions de recherche similaires et décrivent les différentes perspectives et les cadres pour évaluer la maturité des LPL. Ils distinguent également les questions en

fonction de l'expertise de la personne interrogée au sein de l'entreprise. Ils discutent de la nécessité d'une approche systématique avec un large éventail de facteurs pour évaluer l'impact sur l'organisation. Ils ont également évalué l'intérêt du Framework d'Évaluation des Familles *(FEF)* [vdLSR07]. Rincón et al. [RMS18, RMS19] ont proposé le framework APPLIES pour décider de la préparation à l'adoption des LPL. Ils fournissent également une évaluation de la perception de ce framework. Ils posent des questions pour décider de la faisabilité d'une migration, tandis que nos questions de recherche correspondent à l'étape où la migration a déjà été décidée. L'une des questions de recherche d'APPLIES fonctionne comme une phase en amont du processus de migration, en se concentrant sur l'évaluation de la motivation et de la préparation d'une organisation à adopter une approche LPL. Dans les premières étapes de la mise en œuvre d'une LPL, divers facteurs façonnent les attitudes des individus envers cette approche. Rincón et al. divise le concept de « confort »en motivation et préparation, s'alignant sur les activités des étapes initiales du processus de décision en matière d'innovation (définition de l'agenda et correspondance). Hetrick et al [HKM06] proposent des solutions pour éviter les barrières de résistance aux projets innovants, tout comme nous. Ils détaillent comment déclencher une réaction en chaîne conduisant au succès de la migration sur un grand projet. Ces articles ont été des sources d'inspiration pour les questions que nous avons conçues. De plus, ils complètent notre travail, car nous pouvons les utiliser pour fournir des stratégies visant à améliorer l'efficacité de l'adoption des LPL. Dans la littérature, il existe des rapports sur l'adoption et la mise en œuvre des techniques de LPL. Les auteurs de [BMK+02] décrivent une méthode utilisée pour passer d'un processus de développement standard à un processus de LPL. Une étude des forces, des avantages et des problèmes rencontrés lors de l'adoption des LPL est présentée dans [vdLSR07]. La motivation d'adopter les LPL dans un contexte agile est développée dans [FSK+16]. Des cas d'adoption de LPL et les méthodes utilisées sont détaillés dans [BNR+14]. De plus, une discussion des facteurs clés (par exemple, les processus, les politiques et les ressources) a été développée pour montrer comment ils influencent l'établissement des LPL. Une enquête évaluant la modélisation de la variabilité dans la pratique industrielle en utilisant l'ingénierie des Lignes de Produits Logiciels [BRN+13] a étudié comment combler le manque de données empiriques sur la validation des techniques existantes. Leur enquête avait pour objectif d'identifier les pratiques, les opportunités et les défis liés à la modélisation de la variabilité et a reçu 67 réponses d'entreprise. À partir des réponses, les auteurs discutent en particulier de l'importance de la modélisation de la variabilité dans l'industrie pour améliorer la qualité et l'efficacité.

Conception des interviews Pour créer nos questions, nous avons utilisé un vocabulaire standard sur lequel nous avons aligné les termes des interviewés. Pour établir ce vocabulaire, nous avons utilisé des termes issus de [MZB+15, Kru92, vdLSR07]. Martinez et al [MZB+15] proposent une approche pratique et efficace pour les organisations intéres-

sées par l'adoption de méthodes de LPL dans leurs pratiques de développement logiciel. Ils se concentrent sur l'évolution d'un ensemble de logiciels en introduisant des fonctionnalités et des pratiques de lignes de produits. Ils appellent cette approche une approche *bottom-up* en opposition à l'approche *top-down* utilisée pour construire une LPL depuis zéro. Nous avons également enrichi notre étude avec l'analyse TOWS [Wei82] héritée de l'analyse SWOT [HLV99] pour aider à la prise de décision et à la planification du projet. La matrice SWOT consiste à répertorier les forces, les faiblesses, les opportunités et les menaces dans quatre quadrants dédiés. La matrice TOWS consiste à identifier les relations entre ces facteurs. Ces deux analyses sont utilisées pour développer des stratégies plus efficaces et éclairées en tenant compte des forces et des faiblesses du projet. Nous avons également conçu nos questionnaires de manière à inclure les acteurs, comme le recommandent (1) Pikkarainen et al [PHS+08], qui expliquent les impacts des méthodes agiles telles que SCRUM ou eXtreme Programming sur la communication, notamment dans le cadre d'un projet de grande envergure ; et (2) McHugh et al [MCL12], qui ont étudié la confiance au sein d'une équipe agile. Les équipes agiles doivent travailler en autonomie et faire face à de nouvelles difficultés. Les deux références [PHS+08, MCL12] soutiennent que les membres de l'équipe agile sont essentiels à la réussite du projet. Comme cela a été démontré dans [MDD08], identifier les risques et insister sur les avantages est bénéfique pour encourager les équipes. Nous avons donc introduit les questions de recherche et des parties du questionnaire dédiées à ces sujets. Ces travaux abordent les facteurs contribuant à l'efficacité des équipes auto-organisées, tels que la présence d'une vision partagée claire ou d'une gestion axée sur l'auto-organisation de l'équipe plutôt que sur le micro-management. Les avantages et limitations les plus significatifs de l'agilité dans le développement logiciel sont détaillés dans [SP16], tels que l'importance de la confiance ou de la satisfaction des employés. Les auteurs fournissent un cadre pour hiérarchiser les avantages et les limitations du développement logiciel agile dans la pratique. Ce cadre consiste à identifier les pratiques agiles, évaluer leur utilisation, à identifier les avantages et les limitations, et à les hiérarchiser en fonction de leur importance.

Agilité Un point de vue sur les défis de gestion pour les équipes agiles dans un cas industriel, développé dans [JNL16], nous a inspiré dans la partie discussion (par exemple, l'adaptation du travail pour améliorer l'implication dans le projet). Les auteurs ont étudié les défis auxquels sont confrontées les petites et moyennes entreprises (PME) au Royaume-Uni lors de l'adoption de pratiques DevOps. Nous avons utilisé leurs lignes directrices (par exemple, une gestion et une organisation efficaces) pour une adoption efficace de nouvelles pratiques. Notre travail est basé sur le développement axé sur l'agilité [BBvB+01], notamment dans les PME [dSdMSNO+14]. Silva et al. [dSdMSNO+14] ont axé leurs recherches sur le cadrage des lignes de produits pour définir les limites, les points communs et les variations dans le cas des PME. Ils cherchent à identifier les défis

auxquels les PME sont confrontées, tels que les conflits entre parties prenantes ou les contraintes de ressources. Comme eux, nous cherchons à identifier les stratégies pour surmonter ces défis, notamment par le biais de processus formalisés et d'investissements dans l'équipe de développement.

Génie logiciel empirique Les auteurs des fondements du Génie Logiciel Empirique discutent de l'héritage de Victor R. Basili, un pionnier de l'ingénierie Logicielle Empirique [BRZ05]. Leur travail met en évidence l'importance des études empiriques comme la nôtre, notamment l'utilisation de données et de preuves pour orienter les pratiques de génie logiciel.

Comme discuté précédemment, de nombreux travaux dans la littérature ont abordé les défis de l'adoption de l'ingénierie des LPL dans un contexte industriel et son impact sur les pratiques de développement agile actuelles. Ces travaux ont proposé de nombreuses procédures intéressantes à suivre et des stratégies à adopter. Ils nous ont donné de bonnes indications sur la manière de procéder dans le processus de migration.

Aucun d'entre eux n'a entrepris une étude d'évaluation empirique similaire à la nôtre, en particulier car la nôtre se déroule après l'adoption et avant la mise en œuvre.

3.8 Conclusion

Dans ce chapitre, nous avons présenté la première étape d'un processus que nous menons pour accompagner une entreprise dans la migration de sa base de code vers une Ligne de Produits Logiciels (LPL). Nous avons réalisé des entretiens que nous avons conçus en tenant compte des spécificités de l'entreprise, et en particulier des différents types de rôles des acteurs du projet. Notre objectif étant d'identifier des orientations pertinentes pour préparer la migration, nous nous sommes concentrés sur des questions de recherche qui nous aideront à connaître les bonnes pratiques actuelles sur lesquelles nous pourrions construire la migration, les avantages et les risques perçus par les acteurs, ainsi que les craintes qu'ils expriment. Proposer des réponses adéquates aux risques et aux craintes, en s'appuyant sur les avantages perçus et en promouvant les bonnes pratiques, sera très précieux pour une transition en douceur. Nous avons identifié que l'entreprise a de bonnes habitudes qui faciliteront le processus et qu'elle a une culture de l'innovation qui sera favorable pour le projet. La majorité des acteurs voient des avantages dans la migration. Enfin, les principaux risques sont la complexité de la LPL et sa maintenance.

Dans la suite de cette thèse, nous allons décrire un processus pour l'analyse des schémas de données d'entrée et de sortie des simulateurs (composants boîtes noires développés par des agronomes) afin de capturer la variabilité. Puis nous présenterons une méthode d'analyse des problèmes liés aux récits utilisateurs dans la plateforme de gestion de version collaborative (Gitlab) de la plateforme afin de : 1) identifier les caractéristiques et

leurs demandes de fusion associées, les messages de commit et les différences du Git, puis 2) localiser ces caractéristiques. Nous exploitons l'Analyse de Concepts Formels [GW99] et l'Analyse de Concepts Relationnels [HHNV13] pour construire le modèle de caractéristiques de la LPL.

La suite se concentrera sur l'étude de la base de code et le processus de développement de l'entreprise afin de proposer un plan de migration : analyse de la base de code, choix d'un type de LPL (annotative, compositionnelle ou hybride), mise en œuvre et application de la technique d'extraction, conception d'une stratégie de maintenance conforme (transparente) aux pratiques actuelles de l'entreprise, parmi d'autres tâches.

CHAPITRE 4

Extraction de variabilité à partir de schémas de données d'entrée/sortie de simulateur dans un logiciel d'aide à la décision pour l'agriculture

Sommaire

Ce chapitre détaille l'étude du développement de systèmes logiciels visant à aider à la prise de décision dans le domaine de l'agriculture auprès de notre partenaire industriel, ITK. Ces systèmes logiciels comprennent des simulateurs qui aident les agriculteurs à

comprendre et à prédire le cycle de vie des plantes. Chaque plante et chaque type de prédiction ont leurs propres paramètres. Par exemple, la prédiction du rendement du blé est très spécifique et différente de la prédiction des maladies de la vigne. Cependant, il y a des caractéristiques communes, comme le fait que ces simulateurs prennent en entrée des données météorologiques. L'objectif du projet sur lequel nous travaillons est de construire une ligne de produits logiciels afin de : i) permettre une dérivation facile de nouveaux produits (par les équipes IT) avec de nouveaux simulateurs (construits par les équipes d'agronomes), et ii) simplifier la maintenance de la grande base de code existante de notre partenaire industriel. La construction de cette ligne de produits passe par l'extraction des caractéristiques variables et communes de tous les produits existants chez ITK. Le processus d'extraction peut être laborieux et chronophage. Nous étudions dans ce travail l'automatisation de ce processus, en nous concentrant sur les schémas de données reçus en entrée et produits en sortie par les simulateurs. Nous émettons l'hypothèse que l'Analyse Formelle de Concepts (AFC) est un outil utile pour extraire la variabilité logicielle [Car18a], c'est-à-dire mettre en évidence les points communs et spécifiques pour aider les équipes IT/agronomes dans la construction logicielle. Dans ce chapitre, nous proposons un processus d'extraction de la variabilité des schémas de données. Ce processus repose sur un ensemble d'étapes de pré-traitement pour préparer les données aux outils AFC. Ces outils construisent à la fin du processus un AOC-Poset, qui est un sous-ordre dérivé du treillis de concepts dans lequel nous pouvons identifier les caractéristiques communes et variables et leur distribution au sein des simulateurs. Nous avons mis en œuvre ce processus et l'avons expérimenté sur un ensemble de six simulateurs. Nous avons obtenu des résultats prometteurs pour la construction de la ligne de produits logiciels.

4.1 Introduction

De nombreuses entreprises réalisant des logiciels sont confrontées au problème de développer et de maintenir une gamme de produits ayant un objectif et un contexte communs. Capitaliser les connaissances acquises dans le domaine et sur les logiciels précédemment développés peut être utile pour développer de nouveaux produits dans le même domaine d'activité et réutiliser les différentes ressources liées à leur développement. Lorsque les systèmes logiciels sont suffisamment similaires, il peut être approprié d'adopter le paradigme de la ligne de produits logiciels afin de capitaliser ces connaissances. Par exemple, notre partenaire industriel ITK [1] propose une plateforme de logiciels d'aide à la décision pour l'agriculture. La plateforme est considérée comme une gamme de logiciels similaires, ce qui peut permettre une migration vers une ligne de produits logiciels. Cela nécessite d'extraire et d'organiser les connaissances à partir de la base de

1. https ://www.itk.fr/

code et de tous les documents existants. Chaque logiciel intègre des simulateurs, développés par les agronomes modélisateurs, par exemple pour estimer les rendements ou la prédire les maladies. Les informations à transmettre et à recevoir lors de la communication avec un simulateur sont fournies par des schémas de données d'entrée et de sortie dans une structure arborescente et hiérarchique des données. Ce chapitre propose une méthode basée sur ces schémas de données d'entrée et de sortie afin d'assister l'équipe agronomique dans le développement d'un nouveau produit et d'accélérer l'intégration des simulateurs par l'équipe de développement. Nous utilisons l'Analyse Formelle de Concepts (AFC) pour mettre en évidence la variabilité des données d'entrée et de sortie parmi les différents simulateurs. Comme indiqué au début de cette thèse, une gestion explicite de la variabilité offre la capacité d'un logiciel à être configuré, personnalisé, étendu ou modifié pour un contexte spécifique [BC05]. Dans le cas précis de l'étude présentée ici, nous nous intéressons à la variabilité dans les schémas de données des simulateurs. Dans ce chapitre, nous expliquons d'une part le processus qui nous mène des données brutes à une structure conceptuelle, ici un AOC-Poset, et d'autre part comment l'exploiter. Concrètement, la contribution de ce chapitre est une application de l'AFC pour identifier la variabilité dans les schémas de données. Le chapitre est organisé de la manière suivante. La section 4.2 donne un aperçu de l'approche. La section 4.3 présente le jeu de données. La section 4.4 explique comment le pré-traitement est effectué sur les schémas de données. La section 4.5 présente le traitement des données par l'AFC. La section 4.6 présente les résultats. La section 4.7 expose les travaux connexes et la section 4.8 conclut le chapitre en résumant la contribution et en proposant quelques perspectives.

4.2 Aperçu de l'approche

Question de recherche La principale question étudiée dans ce chapitre est la suivante : *Comment extraire la variabilité logicielle à partir des schémas de données des simulateurs ?* Pour répondre à cette question, nous utilisons les schémas de données des simulateurs pour construire un contexte formel utilisable avec l'analyse formelle des concepts (AFC), et plus précisément les algorithmes de construction de l'AOC-Poset. Cela est réalisé par un processus, présenté dans la sous-section suivante, qui est capable d'exploiter les schémas de données des simulateurs pour obtenir un certain nombre de connaissances.

Processus La figure 4.1 illustre la méthode permettant d'obtenir la variabilité à partir des schémas de données bruts. Tout d'abord, un pré-traitement est effectué en nettoyant, enrichissant et formattant les données pour obtenir un contexte formel. Ensuite, à partir du contexte formel, une structure basée sur AFC, l'AOC-Poset [BGH+14], est utilisée pour mettre en évidence les points communs et les spécificités des données. Le résultat de cette approche est un modèle partiel de variabilité, avec des termes communs aux simulateurs et des termes spécifiques à certains d'entre eux. L'identification de règles

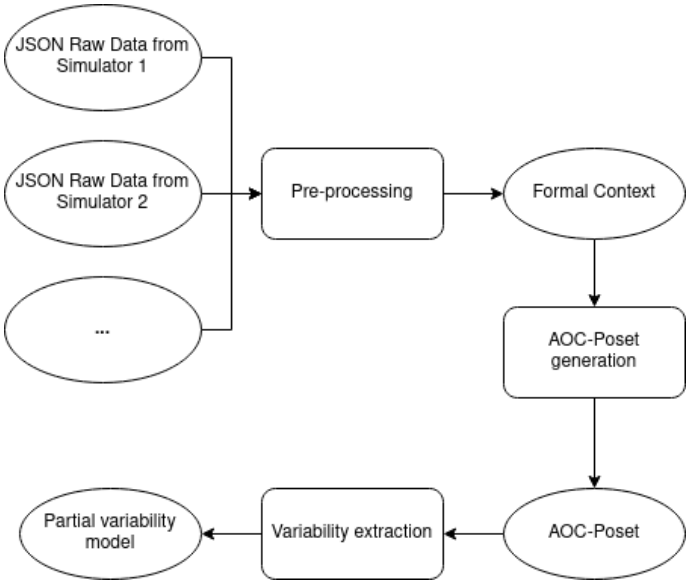


FIGURE 4.1 – Des simulateurs à la variabilité

logiques reliant ces termes, telles que des implications, des co-occurrences ou des exclusions mutuelles entre deux termes, est possible en appliquant des techniques proposées dans des travaux précédents [CHN18].

Une structure d'AOC-Poset a été choisi au lieu d'un treillis de concepts pour deux raisons. Tout d'abord, l'AOC-Poset est un modèle plus simple, qui le rend facilement lisible et compréhensible dans le contexte de l'analyse de la variabilité. La deuxième raison théorique est la complexité de sa construction. Le nombre de concepts est linéaire dans le nombre de lignes et de colonnes du contexte formel pour l'AOC-Poset. Dans la construction des treillis de concepts, le nombre de concepts dans le treillis peut être exponentiel dans le pire des cas. La taille des schémas de données des simulateurs est relativement petite pour le moment dans notre étude, mais à l'avenir, si le processus doit être utilisé avec des schémas plus grands et plus nombreux, la construction du modèle de variabilité serait rendue plus efficace.

4.3 Description des simulateurs

Chaque simulateur utilisé dans les produits ITK a son propre objectif. Il reçoit des données en entrée, telles que des informations météorologiques ou le type de sol. Il produit des données en sortie, telles que des prédictions de rendement ou de maladies. Tous les produits ITK sont définis comme des applications Web avec des simulateurs principalement écrits en Python. Les données d'entrée et de sortie sont définies au format

JSON[2] et les schémas les décrivant sont aussi définis en *JSON*.

Les schémas de données de sortie sont généralement moins complexes que ceux des données d'entrée et peuvent être absents dans certains cas, s'il n'y a qu'un nombre ou une chaîne de caractères en sortie d'un simulateur.

Notre étude est basée sur six simulateurs différents :

- *Cropwin simulator* pour estimer le rendement des cultures annuelles comme le blé ou le maïs.
- *Disease simulator* pour prédire le risque de maladie des plantes.
- *Grapes simulator* pour estimer le rendement des cultures de raisins.
- *nferti simulator* pour prédire le stress des plantes, comme le manque d'azote.
- *Orchard simulator* pour estimer le rendement des cultures durables comme les abricots ou les noix.
- *Vine Disease simulator* pour prédire le risque de maladies spécifiques des vignes.

Description du simulateur Chaque simulateur dispose d'une documentation fournie par les agronomes qui l'ont développé. Cette documentation comprend un wiki avec une description du modèle de simulation (le modèle mathématique), l'API pour utiliser le simulateur, ses dépendances et sa documentation technique. Le modèle de simulation fournit le schéma des données nécessaires pour exécuter le simulateur (données d'entrée) ainsi que le schéma des données fournies en sortie. Ces schémas sont définis au format JSON également. Nous avons collecté les schémas disponibles pour les six simulateurs sélectionnés.

Schéma de données d'entrée/sortie. Chaque schéma de données est un arbre de termes, dont un exemple est donné à la figure 4.2. Parmi ces six simulateurs, nous avons six schémas de données d'entrée et quatre schémas de données de sortie. Les schémas de données d'entrée comprennent de 31 à 127 termes différents et les schémas de données de sortie comprennent de 22 à 95 termes différents (voir le tableau 4.1 pour plus de détails).

Simulateur	nferti	Cropwin	Disease	Grapes	Orchard	Vine Disease	**total**
Termes en entrée	119	126	127	31	50	11	**464**
Termes en sortie	95	32			53	22	**202**

TABLE 4.1 – Nombre de termes décrivant les données en entrée et en sortie des simulateurs

2. https ://www.json.org/json-en.html

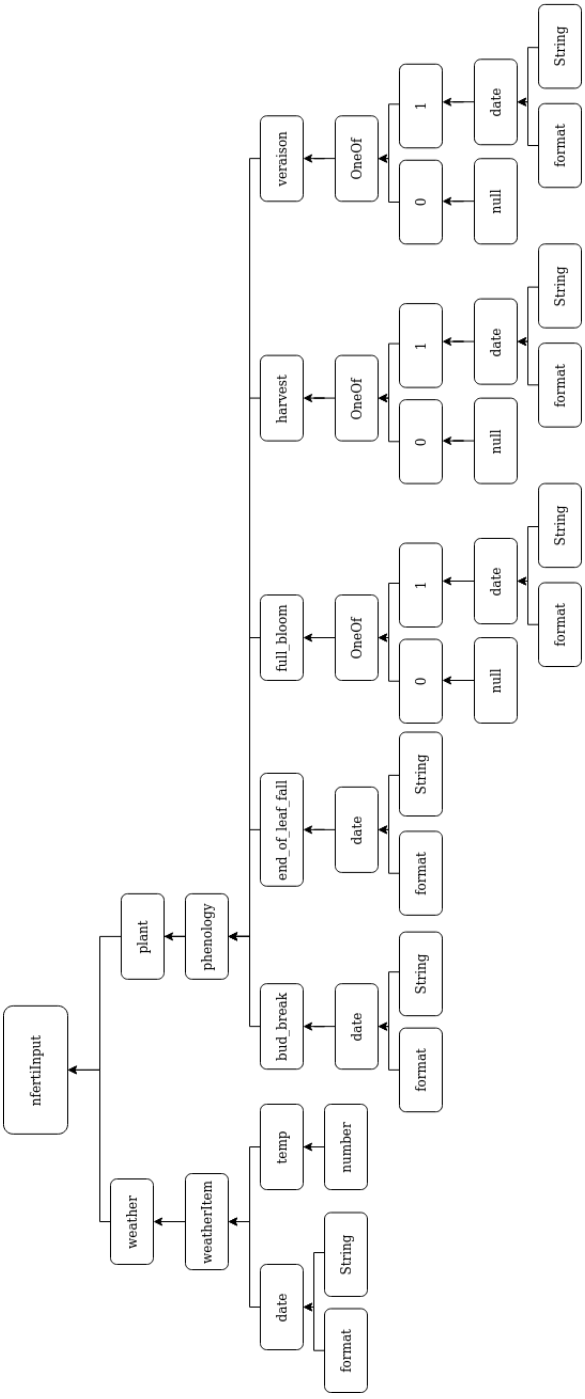


FIGURE 4.2 – Exemple d'une partie d'un schéma d'entrée

4.4 Pré-traitement des schémas de données

Les schémas de données bruts ne peuvent pas être directement exploités dans notre processus. Ils doivent être nettoyés, formatés et préparés pour être exploités par l'AFC. Ces pré-traitements sont détaillés dans cette section : un aperçu du pré-traitement (Sect. 4.4.1), suivi des différentes étapes avec l'exclusion des mots indésirables (Sect. 4.4.2), l'association de nouveaux termes (Sect. 4.4.3), extraction de tuples (Sect. 4.4.4) et le formatage des données (Sect. 4.4.5).

4.4.1 Aperçu du processus de construction du contexte formel

L'AFC se base sur des contextes formels représentant les objets et leurs attributs. À partir des schémas de données, plusieurs pré-traitements ont été appliqués afin d'obtenir un contexte formel représentant les caractéristiques des simulateurs.

Nous avons construit un dictionnaire pour exclure les termes indésirables/techniques, un dictionnaire pour associer de nouveaux termes afin de remplacer les acronymes et les abréviations existants. Afin de maximiser l'extraction de la variabilité, nous choisissons d'exploiter des tuples (n-uplets) de termes. Les résultats ont été formatés sous forme de contexte formel. La construction de nos dictionnaires en supprimant ou en associant davantage de termes a été un travail manuel itératif. Une analyse a été réalisée après chaque itération. Les termes redondants et inappropriés ont été supprimés et de nouveaux termes expliquant les abréviations existantes ont été ajoutés.

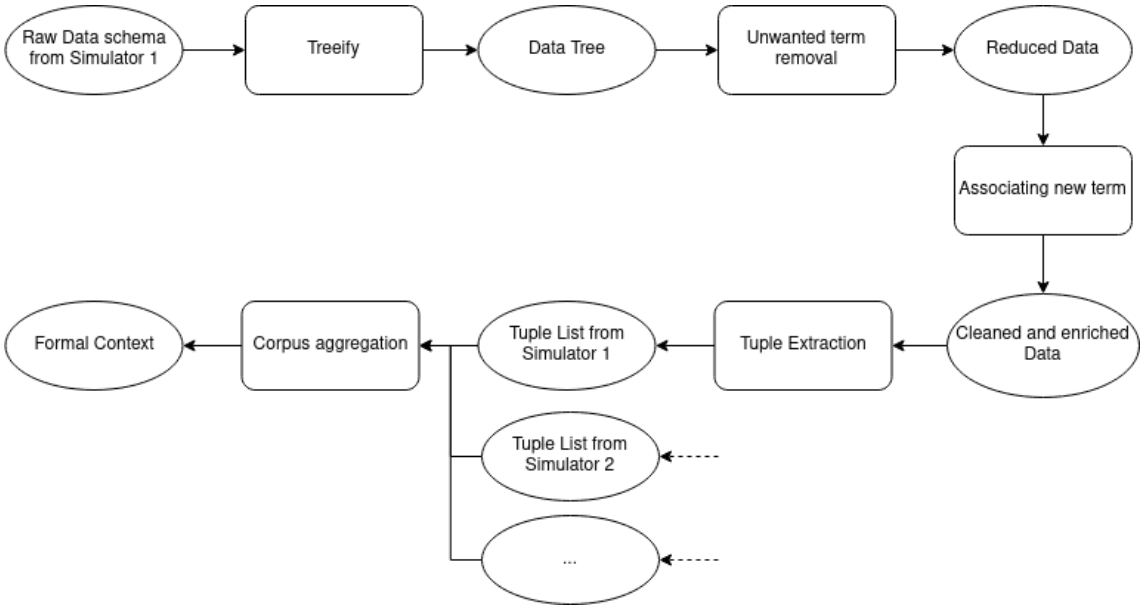


FIGURE 4.3 – Processus de transformation des données brutes en contextes formels.

Dans la figure 4.3, nous présentons le processus complet qui va des données brutes à un contexte formel. Tout d'abord, nous transformons les schémas de données brutes en arbre (*Treeify*) afin de pouvoir effectuer les étapes de traitement suivantes sur un arbre et non sur un document texte. Ensuite, nous supprimons les termes inutiles (*Suppression des termes indésirables*). Après cela, nous remplaçons les abréviations et les acronymes (*Association de nouveaux termes*) et nous extrayons des tuples de l'arbre (*Extraction des tuples*). Enfin, pour tous les simulateurs, les schémas de données de chaque type (entrée ou sortie) sont fusionnés et formatés en tant que contexte formel (*Agrégation du corpus*).

4.4.2 Exclusion des termes indésirables

Les termes indésirables (*stopwords*) sont des termes qui n'ont pas besoin d'être conservés dans le processus d'identification de la variabilité. Nous choisissons de les supprimer pour limiter les termes trop fréquents et inutiles. Sans supprimer les termes indésirables, l'extraction de la variabilité serait moins pertinente en raison du bruit introduit.

Pour construire le dictionnaire, nous avons classé tous les termes par fréquence, afin de sélectionner et de supprimer tous les termes indésirables trop fréquents. Nous avons utilisé une métrique bien connue à cet effet, TF-IDF [Ram03]. Nous avons construit un dictionnaire contenant 30 termes différents à supprimer.

Nous avons par exemple éliminé les termes indésirables suivants : *Type*, *String*, *definitions* ou *properties*.

Ces termes ont été identifiés à partir des données. Bien que des dictionnaires existent, aucun ne correspondait spécifiquement à nos données. Cependant nous avons utilisé plusieurs ontologies obtenues depuis la plateforme AgroPortal [3] afin de confirmer la pertinence des termes.

La figure 4.4 illustre un exemple de ce traitement. Chaque terme dans un schéma de données est recherché dans le dictionnaire des termes indésirables et supprimé de l'arbre s'il est présent (par exemple, *type* et *string*). Si le terme supprimé n'est pas une feuille de l'arbre, le sous-arbre lié à ce nœud supprimé est directement lié à son parent.

4.4.3 Association de nouveaux termes

L'objectif de la construction de ce deuxième dictionnaire est d'étendre les acronymes et de remplacer les abréviations par les termes complets. La construction de ce dictionnaire a été réalisée manuellement. Nous avons vérifié chaque terme pour décider s'il s'agissait d'un acronyme ou d'une abréviation. Dans ce cas, nous l'avons ajouté au dictionnaire avec son nom complet. Par exemple, *irrig* a été ajouté et associé au terme *irrigation*. Le dictionnaire construit comprend environ 30 termes différents, dont 9/10 proviennent du domaine agronomique et 1/10 du domaine informatique.

3. https://agroportal.lirmm.fr/

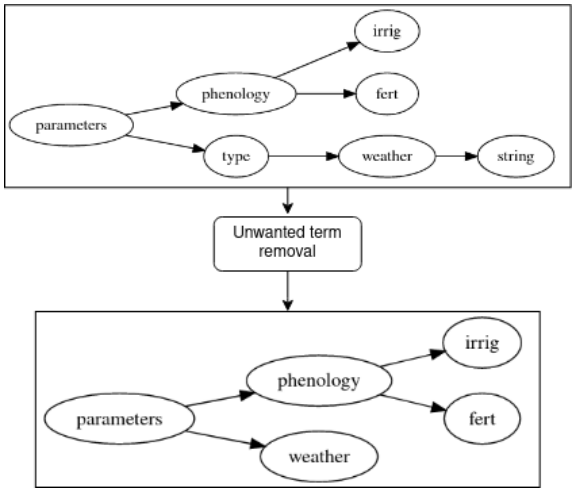


FIGURE 4.4 – Exemple de suppression de termes indésirables

TABLE 4.2 – Exemples d'associtions

abreviation	association
temp	temperature
pheno	phenology
c	carbon
n	nitrogen
irrig	irrigation
simu	simulator
sim	simulator

Le tableau 4.2 montre quelques exemples de ces associations.

Nous avons aussi validé ces termes en utilisant des ontologies du domaine pour valider le vocabulaire spécifique au domaine des simulateurs ITK, pratiqué par ses agronomes.

La figure 4.5 montre un exemple de ce traitement. Chaque terme est comparé au terme associé dans le dictionnaire. S'ils correspondent, le terme actuel est remplacé par sa version complète.

4.4.4 Extraction des tuples

L'extraction de termes uniques nous fait perdre les informations sur les relations fournies par la structure de l'arbre.

Pour conserver les informations provenant des données, nous devons affiner l'extraction. Nous extrayons chaque nœud individuellement, mais aussi toutes les paires père-fils

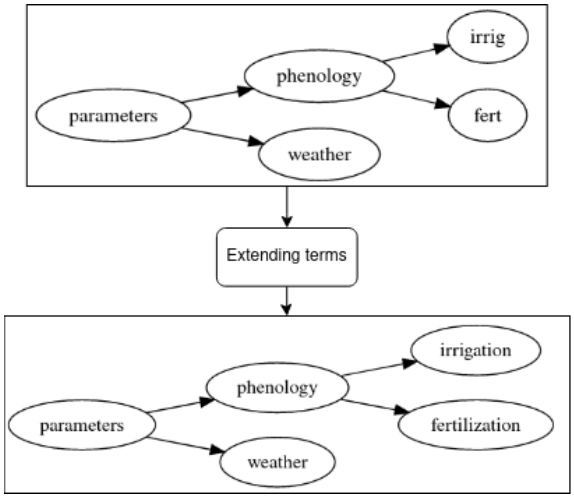


FIGURE 4.5 – Exemple d'extension de termes

en suivant une méthode de recherche en profondeur. La taille des tuples extraits est d'un ou deux termes, ce qui est suffisant pour notre processus. En effet, après des observations empiriques, nous avons conclu que l'utilisation de tuples de trois termes ou plus n'aide pas à identifier davantage de similarités dans nos données, car ils ne sont présents que dans un seul simulateur. Par exemple, le tuple `(parameters, phenology, irrigation)` est spécifique au seul simulateur *nferti*.

De plus, nous n'avons pas basé notre processus sur des données textuelles brutes et avons choisi de les transformer en notre propre représentation arborescente, afin d'être indépendants de tout type de format de structure de données, tel que XML ou JSON dans notre cas.

La figure 4.6 illustre un exemple d'extraction de tuples. Dans cet exemple, nous pouvons observer l'extraction de cinq tuples avec un seul terme et quatre tuples avec deux termes, à partir d'un arbre de quatre nœuds et de trois feuilles.

4.4.5 Formatage des données

Pour utiliser l'AFC, une dernière transformation est requise : la transformation en contextes formels.

La figure 4.7 présente un exemple de formatage des données brutes en contexte formel. Nous avons deux ensembles, G et M, qui représentent les simulateurs et les attributs, respectivement.

Nous avons ensuite utilisé l'AFC comme méthode d'ingénierie des connaissances, pour sa capacité à construire des *concepts formels* à partir d'un contexte formel (CF) $K = (G, M, I)$ qui associe des objets d'un ensemble G à des attributs d'un ensemble M par le biais de la relation $I \subseteq G \times M$. Les ensembles d'objets et d'attributs sont associés

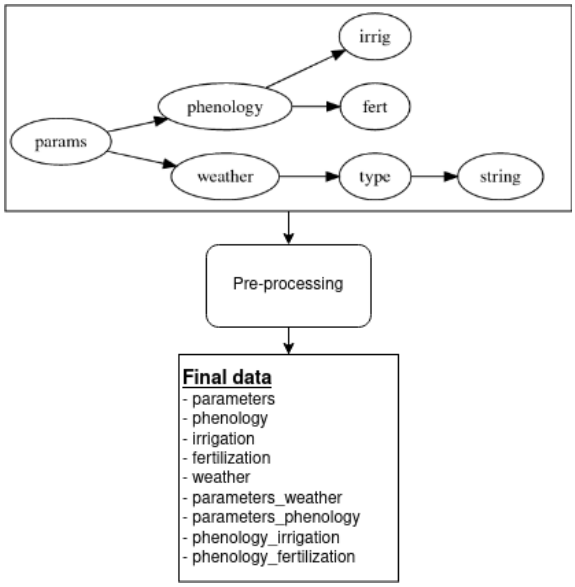
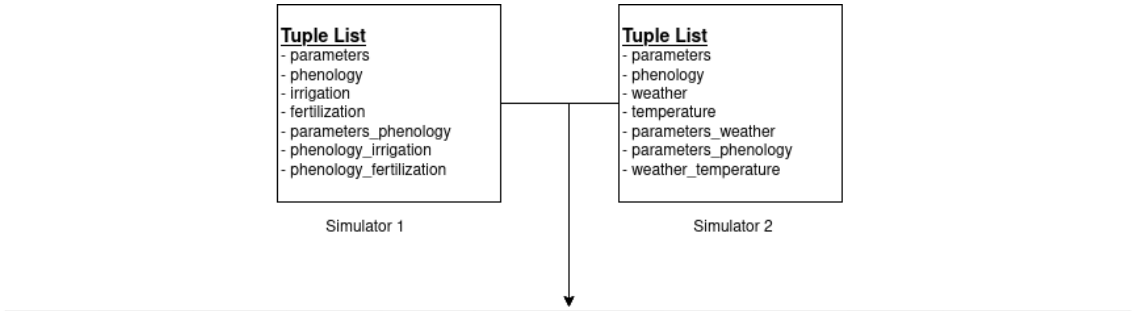


FIGURE 4.6 – Des données nettoyées et enrichies à des attributs formels (colonnes du contexte formel qui va être construit).

grâce à deux opérateurs, tous deux notés par $'$. Pour $O \subseteq G$, l'ensemble des attributs partagés par les objets de O est $O' = \{m \in A | \forall g \in O, (g, m) \in I\}$. Pour $A \subseteq M$, l'ensemble des objets qui partagent les attributs de A est $A' = \{g \in G | \forall m \in A, (g, m) \in I\}$. Un concept formel $\mathcal{C} = (Extent(\mathcal{C}), Intent(\mathcal{C}))$ est un groupe d'objets maximal (extension) associé à son groupe d'attributs partagés maximal (intension [4]), c'est-à-dire $Extent(\mathcal{C}) = Intent(\mathcal{C})'$ (et de manière équivalente $Extent(\mathcal{C})' = Intent(\mathcal{C})$). L'ordre des concepts, noté $\preceq_{\mathcal{C}}$, est défini comme suit : $\mathcal{C}_1 \preceq_{\mathcal{C}} \mathcal{C}_2$ si $Intent(\mathcal{C}_2) \subseteq Intent(\mathcal{C}_1)$ (et de manière équivalente $Extent(\mathcal{C}_1) \subseteq Extent(\mathcal{C}_2)$). Le treillis de concepts est l'ensemble de tous les concepts muni de l'ordre partiel $\preceq_{\mathcal{C}}$. Le concept le plus bas (par rapport à $\preceq_{\mathcal{C}}$) possédant un objet est son concept introducteur. Le concept le plus élevé (par rapport à $\preceq_{\mathcal{C}}$) possédant un attribut est son concept introducteur. Le sous-ensemble du treillis de concepts restreint à ces concepts introducteurs est appelé l'AOC-Poset (Attribute-Object Concept partially ordered set). Dans la suite, nous utilisons les AOC-Posets, qui sont une alternative aux treillis de concepts, en tant que structures conceptuelles, pour mettre en évidence la variabilité, car ils contiennent toutes les informations dont nous avons besoin et sont plus compacts.

Nous avons généré deux contextes formels à partir des tuples extraits : l'un pour les schémas de données d'entrée et l'autre pour les schémas de données de sortie. Dans chaque contexte formel, G est l'ensemble des simulateurs et M est l'ensemble des tuples, c'est-à-dire les attributs extraits. La relation I est une relation binaire qui indique si un

4. L'orthographe de ce terme est particulière ; le terme provient du vocabulaire de la philosophie.



Terms	parameters	phenology	irrigation	fertilization	weather	temperature	parameters_ weather	parameters_ phenology	phenology_ irrigation	phenology_ fertilization	weather_ temperature
Simulator 1	x	x	x	x				x	x	x	
Simulator 2	x	x			x	x	x	x			x

FIGURE 4.7 – Exemple de données transformées en contexte formel

simulateur possède ou non un attribut.

Le contexte formel des entrées est composé de 6 objets (un par simulateur) et de 410 attributs. Le contexte formel des sorties est composé de 2 objets de moins car les sorties des simulateurs *Grapes* et *vintel_desease* ne sont pas spécifiées. Cela fait au final 4 objets et 198 attributs.

4.5 Utilisation de l'analyse formelle de concepts

Les données nettoyées sont maintenant dans le format approprié pour être traitées par les outils de l'AFC afin d'en extraire les informations de variabilité.

Nous avons utilisé le plugin ARC de Cogui [5] pour générer les AOC-Poset (Attribute-Object Concept) [DLBH13]. Pour les schémas de données d'entrée, nous avons obtenu l'AOC-Poset représenté dans la figure 4.8. Cette structure conceptuelle est discutée dans la section suivante.

5. http ://www.lirmm.fr/cogui/

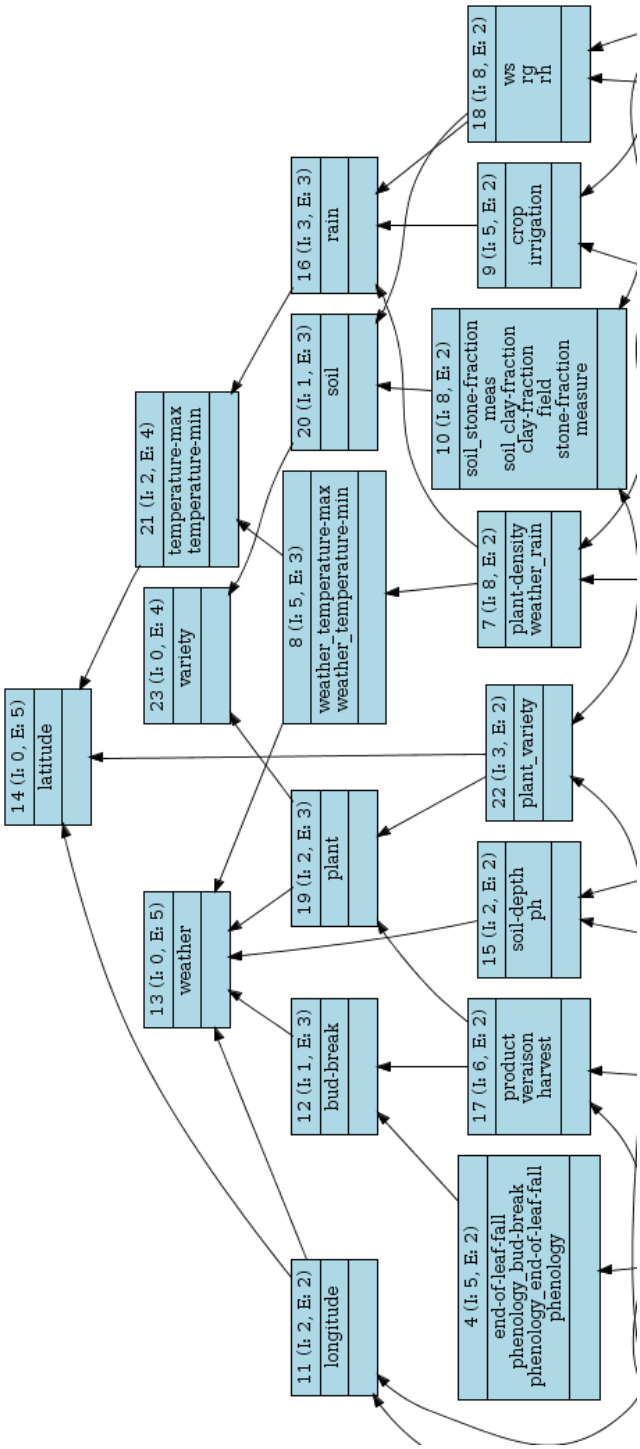


FIGURE 4.8 – Extrait de l'AOC-Poset construit à partir des schémas de données d'entrée.

4.6 Évaluation

La variabilité extraite à partir des AOC-Posets est utilisée pour comprendre dans quelle mesure les simulateurs sont proches les uns des autres.

4.6.1 Résultats

L'extrait de l'AOC-Poset présenté dans la figure 4.8 montre quelles sont les parties spécifiques et communes des simulateurs. Plus précisément, en commençant par les parties les plus souvent partagées (à partir des concepts les plus hauts de l'AOC-Poset), nous avons :

— Le concept 14 introduisant l'attribut *latitude*, commun à cinq simulateurs sur six.

— Le concept 13 introduisant l'attribut *weather*, commun à cinq simulateurs sur six.

— Le concept 23 introduisant l'attribut *variety*, commun à quatre simulateurs sur six.

— Le concept 21 introduisant les attributs *temperature-max* et *temperature-min*, communs à quatre simulateurs sur six.

La *température* et la *météo* ont un impact significatif sur les cultures, chaque *variété* a sa propre spécificité et la *latitude* est utile pour l'impact de l'ensoleillement. En regardant dans le domaine agronomique, on peut facilement comprendre pourquoi ce sont les termes les plus courants.

Nous avons 36 termes communs à au moins deux simulateurs et 374 termes spécifiques sur 410 au total. Cela signifie que les simulateurs ont huit pour cent de termes en commun, tous les autres concepts n'étant présents qu'une seule fois, spécifiques à leur propre simulateur. La partie inférieure de l'AOC-Poset n'est pas affichée dans la figure 4.8 car elle contient des informations inutiles, qui correspondent à six concepts qui correspondent aux six simulateurs et à tous leurs attributs spécifiques (non communs avec les autres simulateurs). Nous pouvons observer que le regroupement des attributs dans les concepts De l'AOC-Poset est variable et varie de un, dans les concepts supérieurs, discutés ci-dessus, à sept dans le concept 10 de la figure 4.8. Nous pouvons également observer que ce dernier concept correspond à une abstraction des simulateurs qui traitent les informations sur le sol, c'est-à-dire qu'il existe une cohésion sémantique entre ces sept attributs. Le concept 4 (à gauche de la figure) regroupe également quatre attributs sémantiquement liés, qui correspondent à la phénologie des plantes.

Pour les sorties nous avons trouvé seulement trois termes communs :

— Le concept introduisant l'attribut *daily*, commun à trois simulateurs sur quatre.

— Le concept introduisant l'attribut *phenology*, commun à trois simulateurs sur quatre.

— Le concept introduisant l'attribut *yearly*, commun à trois simulateurs sur quatre.

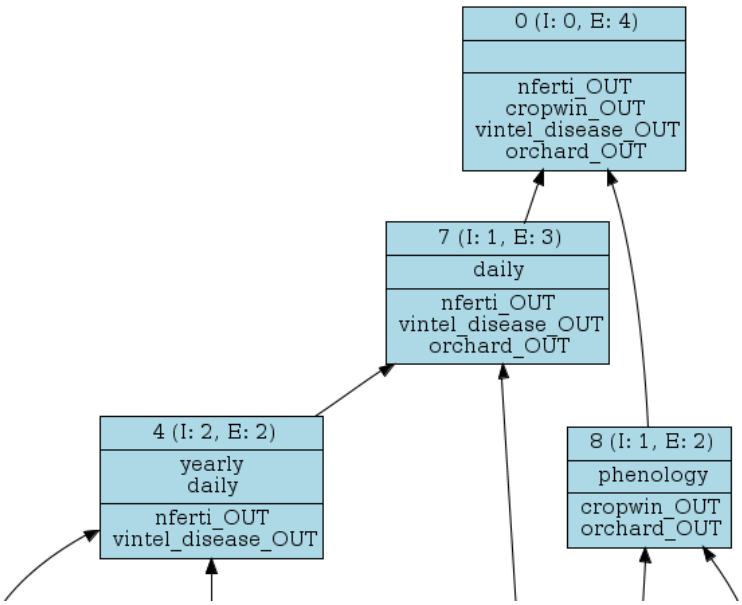


FIGURE 4.9 – Treillis de concepts construit à partir des schémas de données de sortie.

Nous avons observé qu'il n'y a que trois termes communs à au moins deux simulateurs et 195 termes spécifiques sur 198 au total. Cela signifie que les simulateurs n'ont qu'un pour cent de termes en commun, tous les autres concepts n'étant présents qu'une seule fois. Ce faible degré de proximité était prévisible car chaque simulateur a sa propre finalité et ne renvoie que des données utiles ciblant cette finalité.

4.6.2 Discussion

Le premier objectif de ces travaux est d'aider les agronomes dans la conception de nouveaux modèles et simulateurs, ainsi que de soutenir l'intégration des simulateurs par l'équipe informatique et de permettre une meilleure migration vers une ligne de produits logiciels. Les méthodes utilisées et les résultats ont été présentés et discutés avec les équipes d'ITK. Les pré-traitements en particuliers ont été partagés aux agronomes et aux développeurs.

Les résultats de ces travaux permettent en outre de fournir des dictionnaires avec du vocabulaire standard. De plus, le dictionnaire d'association aide à comprendre quels termes sont utilisés, y compris les acronymes, les abréviations ou les écritures différentes, et offre une nouvelle normalisation. Nous sommes assez confiants que cela aidera les agronomes lorsqu'il s'agit de construire un nouveau modèle en fournissant un

vocabulaire standardisé et des conventions de dénomination.

Lorsque l'équipe informatique doit développer une nouvelle application, sa première tâche est l'intégration du simulateur. Il s'agit d'une tâche fastidieuse et sujette aux erreurs, basée sur la copie manuelle d'une intégration existante. En se basant sur la variabilité générée, qui sera liée aux artefacts de code, cette copie peut être automatisée et simplifiée.

Une utilisation finale est la migration des produits logiciels ITK vers une ligne de produits logiciels, en considérant l'AOC-Poset comme une étape vers la production d'un modèle de caractéristiques et de variabilité complet.

Le chapitre suivant détaille comment la migration sera enrichie par la réalisation de modèles de caractéristiques à partir des récits utilisateurs et du code source.

4.7 Travaux connexes

L'AFC a déjà été utilisée pour l'extraction de la variabilité dans le domaine des lignes de produits logiciels, afin de synthétiser un modèle de caractéristiques en explorant l'AOC-Poset (ordre partiel des Attribute-Object-Concepts, c'est-à-dire des concepts introducteurs d'attributs ou d'objets), l'AC-Poset (ordre partiel des Attribute-Concepts, c'est-à-dire des concepts introducteurs d'attributs) ou des systèmes implicatifs .

L'article de Loesch et al. [LP07a] propose une méthode basée sur l'analyse conceptuelle des configurations de produits pour réorganiser la variabilité. Ryssel et al. [RPK11] se concentrent sur l'extraction automatique de modèles de caractéristiques à partir de contextes formels. Al-Msiedeen et al [AHS+14] proposent une approche d'ingénierie inversée pour extraire des modèles de caractéristiques à partir de configurations logicielles. Carbonnel et al. [CHN19b, Car18b] utilisent des treillis de concepts pour modéliser les classes d'équivalence de modèles de caractéristiques, facilitant ainsi leur extraction à partir de descriptions de produits. Ces articles démontrent l'efficacité de l'analyse conceptuelle dans la gestion de la variabilité et l'extraction de modèles de caractéristiques dans le contexte des LPL.

Dans ces travaux, le contexte formel associe des configurations de produits logiciels à des caractéristiques. Le modèle de caractéristiques est un arbre exposant des caractéristiques obligatoires et optionnelles, des groupes de caractéristiques (OR, XOR) et un raffinement des caractéristiques par le biais des arêtes de l'arbre [KCH+90b]. Des contraintes transversales (telles que des implications binaires ou des exclusions mutuelles) peuvent accompagner la description. Nous avons identifié des indications sur la variabilité en suivant les mêmes principes, en utilisant l'AFC pour mettre en lumière des similarités et des spécificités. En comparaison par rapport aux travaux existants, nous ne nous concentrons pas sur la variabilité des caractéristiques, mais sur la variabilité des données d'entrée/sortie. Cela offre une vue complémentaire sur la future ligne de produits.

Nous avons aussi considéré l'utilisation de l'AC-Poset pour la représentation de la

variabilité des caractéristiques en perdant la factorisation des simulateurs.

Le traitement de la description en arbre aurait pu être abordé en s'inspirant des *genterms* (arbres étiquetés dotés d'une relation de généralisation) [DH98] ou en utilisant le paradigme des *pattern structures* (structures de motifs) [GK01,LBTN15]. Dans notre cas, il n'était pas clair initialement de déterminer un opérateur de similarité pour nos arbres racinés étiquetés. Dans ce travail, nous avons préféré mener une première étude en utilisant des informations locales, basées sur les nœuds et les arêtes communs, qui peuvent être encodées avec des contextes formels. Un inconvénient de notre approche est que les portions d'arbre devront être reconstruites lors d'une opération de post-traitement si nous souhaitons avoir une vue globale, mais cela a le mérite de fournir une solution rapide et efficace.

Enfin, ce travail partage également des objectifs et des techniques similaires avec l'intégration de schémas de bases de données [BLN86], la fusion d'ontologies [SM01] et l'extraction de modèles communs [AGM+12], y compris le besoin d'analyse linguistique et la conception d'une vue intégrée, ici sur les entrées ou les sorties des simulateurs.

4.8 Conclusion

Nous avons proposé une méthode pour l'identification de la variabilité dans les schémas de données des simulateurs afin d'améliorer le développement futur de nouveaux simulateurs et d'assister leur intégration dans un nouveau logiciel.

À court terme, nous devons intégrer ces schémas dans le processus de migration de la ligne de produits logiciels. Tous les simulateurs existants ne disposent pas d'un schéma de description des données, en particulier pour les sorties. Cela entraîne un manque de détails sur la variabilité et réduit l'impact des résultats. L'inclusion de plus de schémas de données provenant d'autres simulateurs et la demande à l'équipe d'agronomes de détailler les sorties lorsqu'elles sont absentes amélioreront la qualité des résultats, nous permettant ainsi de fournir une assistance accrue aux équipes d'ingénierie des LPL. L'utilisation de l'AFC, en particulier de l'AOC-Poset, pourrait être améliorée en cherchant à identifier les évolutions dans le temps et en proposant une visualisation explicite des ajouts, des suppressions et des modifications dans l'AOC-Poset

CHAPITRE 5

Combler le fossé entre les récits utilisateurs et les modèles de caractéristiques en tirant parti des systèmes de contrôle de versions : une étape vers la migration de la gamme de produits logiciels

Sommaire

Dans ce chapitre, nous proposons une méthode de synthèse de modèles de caractéristiques considérant les exigences en agilité et qui exploite différents types de connaissances. Cette méthode est une étape en vue d'une migration de la base de code de notre partenaire vers une LPL. Les connaissances exploitées comprennent les récits utilisateurs, ainsi que des demandes de fusion de code source sur des plateformes de contrôle de versions, pour l'implémentation des récits utilisateurs. Nous considérons les demandes de fusion dans les systèmes de contrôle de versions comme le point central entre les récits utilisateurs (exigences) et le code source (implémentation). Nous exploitons les *ontologies de domaine* pour enrichir et mieux organiser ces connaissances. Nous combinons plusieurs approches pour effectuer la synthèse des modèles de caractéristiques. Le traitement du langage naturel et le regroupement des récits utilisateurs sont utilisés pour identifier les caractéristiques (étape TALN). L'analyse formelle de concepts est utilisée pour les classer de manière hiérarchique (étape FCA). Les règles logiques générées en analysant les résultats des étapes TALN et FCA sont utilisées pour affiner les contraintes portant sur les caractéristiques. Nous avons mis en œuvre et évalué cette méthode sur un ensemble de données provenant de notre partenaire industriel. Les résultats obtenus montre efficacité de notre approche.

5.1 Introduction

Alors que les lignes de produits logiciels (LPL) sont désormais un moyen bien établi de produire efficacement des produits logiciels hautement configurables, la migration d'une famille de logiciels existante vers une plateforme LPL intégrée reste un défi, et les organisations peuvent hésiter à adopter cette approche. Les hésitations découlent souvent du manque de procédures standard guidant le processus, ainsi que des doutes concernant le rapport coût/avantage. Bien que les entreprises possèdent de vastes bases de code bien documentées, elles construisent encore manuellement des applications sur mesure à partir du code de base pour répondre aux exigences de leurs clients, parfois avec une stratégie *clone-and-own* (clonage, c'est-à-dire copie du code, et appropriation, consistant à l'adapter aux nouveaux besoins ou au nouveau contexte).

L'une des difficultés lors du processus de migration est d'extraire un modèle de caractéristiques pertinent à partir des produits existants. Dans ce chapitre, nous abordons cette difficulté dans le contexte de produits construits avec un processus agile, plus précisément lorsque les exigences sont définies à l'aide de récits utilisateurs et qu'un système de contrôle de version (VCS) [Spi05] est utilisé pour stocker et retracer les artefacts du projet. Les récits utilisateurs sont largement utilisés de nos jours dans une grande partie

des projets logiciels pour définir les exigences. Les plateformes de VCS telles que GitLab ou Github sont largement utilisées non seulement pour gérer les artefacts de code, mais aussi comme une partie centrale de la gestion de projet, en utilisant directement des outils intégrés pour gérer les récits utilisateurs ou en intégrant des outils externes (par exemple Jira ou Bitbucket) dans la plateforme de VCS. Par conséquent, notre approche convient à une grande partie des projets modernes et agiles.

Dans ce contexte, nous avons développé une approche automatisée de génération de modèles de caractéristiques basée sur les connaissances collectées à partir des récits utilisateurs et de leur lien avec le code source grâce aux informations de fusion extraites de la plateforme de VCS.

Nous avons implémenté cette méthode semi-automatisée de synthèse de modèles de caractéristiques en utilisant une combinaison de différentes techniques :

— Le traitement du langage naturel (TALN) est utilisé pour analyser les récits utilisateurs et identifier l'ensemble initial de caractéristiques concrètes.

— La vectorisation, le regroupement et l'intégration de connaissances en provenance d'une ontologie prennent en compte l'ensemble initial de caractéristiques concrètes et identifient des caractéristiques abstraites. Ces caractéristiques nous permettent de construire des hiérarchies préliminaires de caractéristiques avec des relations de raffinement.

— L'analyse formelle de concepts (AFC [GW99]) et l'analyse relationnelle de concepts (ARC [HHNV13]) sont appliquées aux résultats précédents. Elles utilisent, en entrée, les liens entre les récits utilisateurs et l'évolution du code source par le biais des fusions de code. Ces liens sont extraits de la plateforme de VCS : ils relient les récits utilisateurs aux fichiers qui les implémentent, en passant par les liens entre les problèmes, les opérations de fusion, les modifications et ensuite les fichiers cibles des modifications. À partir de ces connaissances, AFC et ARC déduisent des groupes interconnectés de produits, de caractéristiques, de rôles et d'artefacts de code source.

— Pour finir, une analyse logique des groupes interconnectés est réalisée afin de déduire des contraintes logiques entre les caractéristiques. Nous avons utilisé ARC et la base de règles d'implication de Duquenne-Guigues [GD86], ce qui nous a permis d'identifier des contraintes à partir des règles.

Nous avons évalué cette méthode sur un ensemble de données construit à partir de la base de code de notre partenaire industriel et de la documentation de leur VCS. Les résultats ont montré que notre approche de synthèse de modèles de caractéristiques utilisant différentes sources de connaissances fournit des modèles de caractéristiques pertinents sur l'ensemble de données industriel. De plus, ces modèles de caractéristiques peuvent être facilement modifiés par un expert pour produire le modèle de caractéristiques réel de la ligne de produits.

Le reste du chapitre est organisé ainsi. La section 5.3 présente un aperçu de l'approche, y compris les artefacts nécessaires. Ensuite, nous détaillons chaque étape de notre processus de synthèse dans la section 5.4, en l'illustrant avec un petit exemple inspiré de l'ensemble de données provenant de notre partenaire industriel. La section 5.5 décrit l'évaluation avec l'ensemble de données de notre partenaire, suivie d'une discussion et d'une analyse des résultats obtenus. Les travaux connexes sont présentés dans la section 5.6. Nous résumons la contribution et tirons des perspectives de ce travail dans la section 5.7.

5.2 Motivation et questions de recherche

5.2.1 Contexte

Dans les projets informatiques modernes, les équipes de développement s'organisent autour de spécifications de projets Agile, telles que les *Epic* et les *récits utilisateurs*. De nos jours, de nombreux projets industriels sont menés en se basant sur cette organisation et tirent parti de l'utilisation de plateformes telles que Github, Gitlab, Jira ou Bitbucket.

L'agilité est un vaste concept, défini par un ensemble de principes. Il n'y a pas une seule « méthode Agile », mais plutôt une variété de cadres et de méthodologies qui s'alignent sur ces principes. L'un des éléments clés de l'agilité est l'implication du client tout au long du processus de développement, favorisant ainsi la collaboration étroite entre les équipes de développement et les parties prenantes externes. Cependant, il est important de noter que l'implication du client n'est pas exclusive des méthodologies Agile. D'autres méthodes de développement logiciel, telles que le modèle en V ou le modèle en cascade, peuvent également intégrer des mécanismes d'interaction client.

Scrum, en particulier, est souvent considéré comme un cadre Agile, plutôt qu'une méthode rigide. Il met l'accent sur l'itération et la flexibilité, avec des cycles de développement courts appelés « Sprints ». Scrum encourage la transparence et l'adaptation et offre une structure de rôles définis, tels que le *Product Owner*, le *Scrum Master* et l'équipe de développement, pour soutenir la collaboration et la gestion du projet. La flexibilité de Scrum permet aux équipes de l'adapter en fonction des besoins spécifiques du projet, ce qui peut entraîner des pratiques légèrement différentes d'une application de Scrum à l'autre. L'agilité offre un cadre flexible et des principes qui favorisent la livraison de produits de haute qualité tout en répondant aux besoins changeants du client.

Parallèlement, les LPL sont introduites avec succès dans un nombre croissant de projets en raison de leurs qualités en matière de développement efficace de produits similaires. La combinaison des approches Agile et du paradigme SPL est un défi qui soulève des questions organisationnelles, techniques et sociales [HMSS17]. L'intégration de méthodes de développement Agile dans les LPL existantes a été envisagée, par exemple, dans [KHPS19], avec des objectifs différents des nôtres comme nous ne verrons par la

suite.

Le contexte de ces travaux repose également sur la collaboration avec le partenaire industriel (ITK – predict and decide [1]) qui développe une famille d'applications similaires pour la prise de décision en agriculture avec une approche Agile. Le partenaire souhaite migrer vers une approche de Ligne de Produits Logiciels (LPL), ce qui constitue un problème inverse par rapport à [KHPS19]. Le partenaire construit ses applications en se basant sur des spécifications rédigées sous forme de récits utilisateurs et sur une base de code bien documentée, gérée sur un serveur Gitlab.

Dans le cadre de ces travaux, nous visons à analyser leur ensemble de récits utilisateurs par produits afin de guider l'automatisation de la synthèse de modèles de caractéristiques [SCW12,She13], dans le cadre d'une étape dans la migration vers une approche SPL.

5.2.2 Questions de recherche

La construction de modèles de caractéristique a plusieurs objectifs. Les vastes logiciels et familles de logiciels peuvent être compliqués à appréhender dans leur globalité. En établissant une cartographie des caractéristiques par rôles et par caractéristiques abstraites, on obtient une vue concrète du système qui peut contribuer à améliorer la documentation du système.

À plus long terme, l'objectif est d'identifier les caractéristiques avec leur implémentation, ce qui initiera la création de variants, et d'avancer dans la mise en place de la ligne de produits logiciels. Nous partons d'une question de recherche générale :

RQ : **Comment les récits utilisateurs d'une famille logicielle peuvent-ils être utilisés pour guider la synthèse de modèles de caractéristiques ?** Cette question se divise en deux sous-questions qui se concentrent respectivement sur l'identification des rôles et des caractéristiques (concrètes), des caractéristiques abstraites et des contraintes logiques. Dans notre contexte industriel, les différents rôles ne partagent pas de caractéristiques concrètes, mais les caractéristiques abstraites peuvent être partagées. Les contraintes logiques doivent être utilisées pour guider la construction d'un modèle de caractéristiques.

RQ1 : **Comment identifier les rôles et les caractéristiques à partir des récits utilisateurs ?** ITK utilise activement les récits utilisateurs comme moyen de spécifier les applications et de maintenir cette spécification à jour. Nous nous basons sur ces récits utilisateurs comme source fiable pour trouver les caractéristiques des applications. La plupart de ces récits utilisateurs se composent de deux parties seulement, décrivant res-

1. https ://www.itk.fr/

pectivement un rôle et la caractéristique écrite du point de vue de ce rôle. Nous émettons l'hypothèse qu'elles correspondent à des caractéristiques au sens des LPL.

RQ2 : **Comment identifier les contraintes logiques qui pourraient guider la synthèse du modèle de caractéristiques ?** Une fois que les caractéristiques sont identifiées pour chaque application et rôle dans cette application, nous disposons d'une description de la variabilité. Cette description contient implicitement les contraintes logiques que nous visons à identifier et structurer. Cependant, un modèle de caractéristiques n'est pas seulement une représentation d'un ensemble de contraintes logiques. Dans ce travail, nous nous concentrons sur l'assistance à un expert pour synthétiser un modèle de caractéristiques en combinant son expertise ontologique et les contraintes logiques identifiées.

5.3 Aperçu de l'approche

La contribution de ce chapitre est une méthode automatisée et extensible de génération de modèles de caractéristiques à partir des ensembles de récits utilisateurs d'une famille de produits.

Le processus est conçu pour se servir de la relation entre les récits utilisateurs et les artefacts du code source ainsi que pour être enrichi par une ontologie. Les relations sont identifiées à partir des fusions de code et à l'aide d'un système de contrôle de versions (VCS) . Nous exploitons le traitement automatique du langage naturel (TALN) et l'analyse relationnelle de concepts (ARC) pour identifier la variabilité et regrouper les caractéristiques. La figure 5.1 illustre le processus avec nos trois entrées : les récits utilisateurs pour chaque produit, une ontologie de domaine et le code source par le biais de l'historique des changements. Par *code source*, nous faisons référence à l'historique des changements de type création, modification et suppression sur les fichiers. Nous utilisons d'abord des techniques de TALN pour identifier les caractéristiques et les rôles et pour classifier les caractéristiques en *clusters*, afin d'obtenir des caractéristiques abstraites. Ensuite, l'ARC regroupe et identifie les contraintes qui sont utilisées pour créer le modèle de caractéristiques, le modèle incluant les rôles et les caractéristiques.

5.3.1 Entrée de la méthode

Les entrées de notre méthode sont les récits utilisateurs par produits, une ontologie de domaine et le code source avec les fusions de code, les changements et les fichiers identifiés avec un système de contrôle de versions.

Une *ontologie de domaine* est un ensemble de concepts et de relations qui décrivent un domaine spécifique. Notre approche se sert de l'ontologie pour nommer les caractéristiques abstraites et améliorer le regroupement. Dans l'état actuel, l'ontologie est utilisée

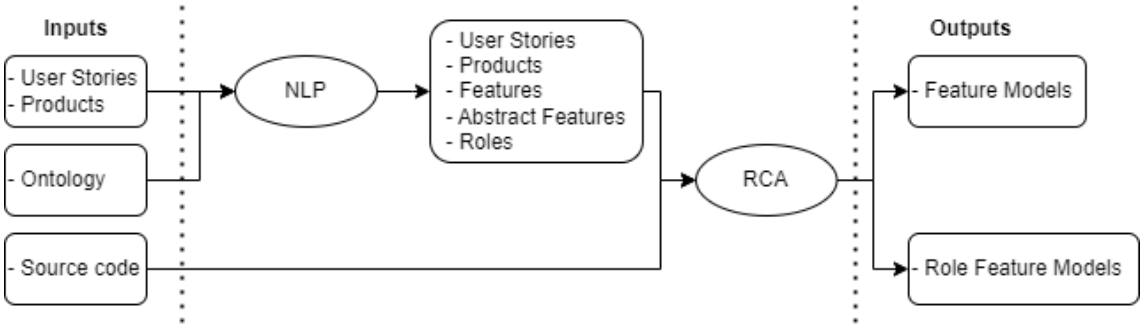


FIGURE 5.1 – Aperçu du processus : à partir des récits utilisateurs par produits, d'une ontologie et du code source nous utilisons le TALN et l'ARC pour déterminer les contraintes et ainsi construire des modèles de caractéristiques et des modèles de caractéristiques par rôles.

manuellement mais n'est pas incluse dans le processus automatique. Elle est également utilisée pour aligner les termes dans les récits utilisateurs lorsque des abréviations ou des synonymes sont utilisés.

Un récit d'utilisateur ou *user story*, est une phrase formatée du point de vue de l'utilisateur, dans le format [Coh04] : « En tant que [rôle], je veux [réaliser cette action], afin d'[atteindre cet objectif] ». Ce format standard est maintenant largement adopté même si différentes variantes existent telles que *Given-When-Then*, *INVEST* ou *Role-Goal-Benefit* qui sont moins courantes.

— Le *rôle* décrit la personne qui a accès à la caractéristique.

— Vient ensuite l'*action* que cet utilisateur doit être en capacité de réaliser.

— La dernière partie décrit l'*objectif* qui est poursuivi par l'utilisateur lorsqu'il réalise cette action.

On peut résumer la syntaxe en « En tant que [rôle], je veux [action], afin [objectif] ». Les récits utilisateurs sont utilisés pour définir les exigences dans les projets agiles. Nous supposons que chaque récit d'utilisateur est associé à un produit au moins.

Une caractéristique [Coh04,NKSB19] est un élément spécifique d'un produit ou d'un système. Nous considérons la partie *[action], [objectif]* du récit utilisateur comme une caractéristique concrète dans nos modèles de caractéristiques.

Le système de contrôle de versions (VCS) qui comprend les fusions de code permet d'identifier les relations entre les exigences et le code source. La figure 5.2 représente l'ensemble des liens au sein du VCS. La figure 5.3 montre comment les récits utilisateurs (nos exigences) sont liées au code source.

Les récits utilisateurs sont supposées être disponibles dans le VCS. Dans notre processus, nous considérons qu'ils sont liés aux *issues*, mais ils pourraient également être stockés dans un outil externe, comme Jira, dès lors qu'ils sont associés aux opérations

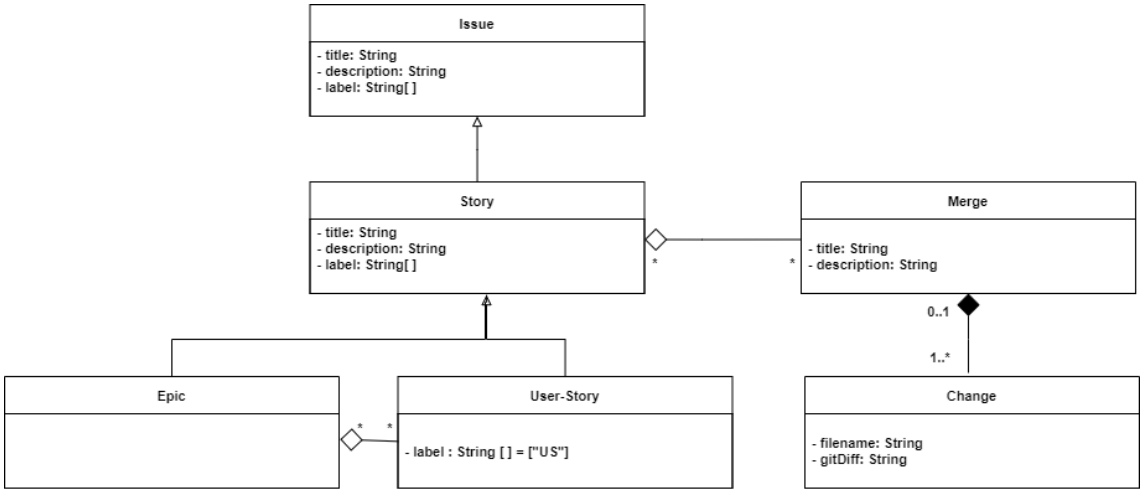


FIGURE 5.2 – Méta-modèle du système de contrôle de versions. Les récits utilisateurs (*UserStory*), super récits (*Epic*) et les récits (*Story*) sont des tickets (*Issue*) du VCS. Les récits agrègent des fusions et les fusions se composent de changements.

de fusion de code dans le VCS. Une *fusion de code* est une opération classique dans les plateformes VCS. Elle correspond à un ensemble de modifications *commit* dans un projet. Elle contient une liste non vide de modifications (création, édition ou suppression) dans les fichiers du projet. Nous supposons que chaque fusion fait référence aux récits utilisateurs auxquelles elle est associée. Nous supposons également que chaque mise en œuvre d'un récit d'utilisateur a fait l'objet d'au moins une fusion de code qui correspond à l'implémentation de la caractéristique. Ces hypothèses correspondent à la réalité dans le processus de développement de notre partenaire industriel.

Le code source est pris en compte afin d'identifier la présence implicite de liens entre les caractéristiques et l'implémentation. Par exemple, observer deux caractéris-

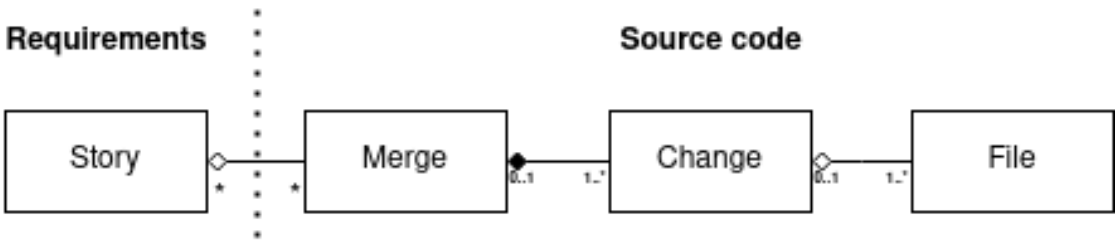


FIGURE 5.3 – Séparation entre les exigences et le code source. Les exigences sont les récits (*Story*) et dans notre cas plus particulièrement les récits utilisateurs. Le code source correspond aux fusions de code (*Merges*), aux changements (*Changes*) et aux fichiers modifiés (*Files*).

tiques ayant des implémentations touchant exactement le même ensemble de fichiers pourrait signifier que ces deux caractéristiques sont liées.

5.3.2 Processus

Traitement Automatique du Langage Naturel (TALN) Le TALN est utilisé en premier lieu pour identifier les rôles et les caractéristiques à partir des récits utilisateurs, puis pour identifier des regroupements de caractéristiques afin de déterminer des caractéristiques abstraites. L'analyse relationnelle de concepts (ARC) est exploitée pour l'extraction des contraintes logiques.

Après l'étape du TALN, nous obtenons les récits utilisateurs, les rôles, les caractéristiques et les caractéristiques abstraites, selon le modèle présenté dans la Figure 5.4.

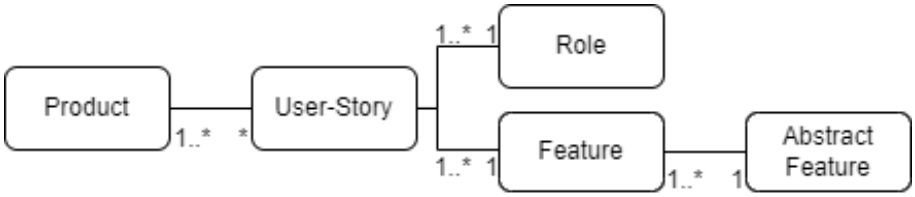


FIGURE 5.4 – Modèle des relations avec les produits liés aux récits utilisateurs. Les récits utilisateurs sont décomposés en rôles et caractéristiques. Les caractéristiques sont catégorisées (regroupées) en caractéristiques abstraites.

La décomposition et la classification des récits utilisateurs sont ensuite utilisées en entrée de l'analyse relationnelle de concepts.

Analyse Relationnelle de Concepts L'apport de l'ARC pour notre méthode est la création de groupes d'objets en exploitant leurs relations. Par exemple, si deux récits utilisateurs concernent le même ensemble de fichiers lors de leur implémentation, les relations vont se propager et ces deux récits utilisateurs seront dans un même groupe associé au groupe de fichiers correspondants. C'est l'ARC qui nous permet d'identifier les contraintes entre les caractéristiques concrètes et abstraites. Par exemple, un concept pourrait regrouper tous les récits utilisateurs partageant le rôle « agriculteur », un autre pourrait regrouper toutes les modifications concernant un groupe de fichiers. Pendant le processus d'ARC, les contextes formels sont étendus avec des attributs relationnels, qui expriment les relations partagées. Par exemple, un attribut relationnel dans la description d'une modification peut exprimer « concerne *au moins* l'un des fichiers du groupe de fichiers X ». Cet attribut relationnel sera attribué à (vérifié par) toutes les caractéristiques concernant au moins l'un des fichiers du groupe de fichiers X.

5.3.3 Sortie de la méthode

Nous construisons deux modèles : le modèle de caractéristiques et le modèle de rôles-caractéristiques.

La racine du *modèle de caractéristiques* est la famille de systèmes analysée, puis viennent les caractéristiques abstraites et enfin les feuilles avec les caractéristiques. Pour chaque caractéristique, nous vérifions s'il s'agit d'une caractéristique obligatoire ou facultative à partir des contraintes identifiés.

Le *modèle de rôles-caractéristiques* est un modèle de caractéristiques qui contient un niveau supplémentaire, décrivant les caractéristiques concrètes et abstraites par rôle. Il contient tous les rôles et les caractéristiques associées telles que décrites dans le modèle de la Figure 5.5.

Les rôles dans le modèle offrent une abstraction supplémentaire avec une vision au-delà des caractéristiques.

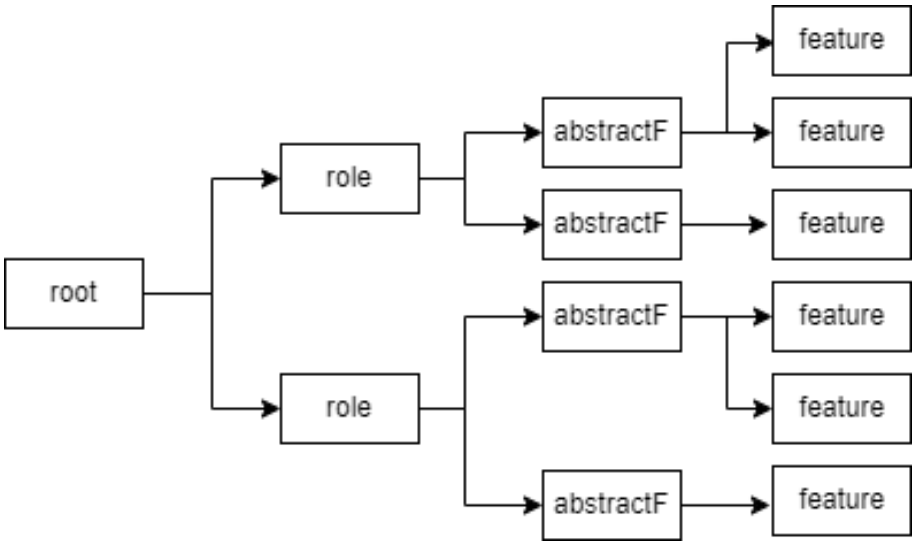


FIGURE 5.5 – Structure du modèle de rôles-caractéristiques. Chaque caractéristique est représentée au niveau le plus bas, et chaque caractéristique possède une caractéristique abstraite parente qui est associée à un rôle.

Outil

Nous avons automatisé le processus proposé en utilisant plusieurs outils. Pour obtenir les entrées de notre processus, nous utilisons d'abord l'*API Gitlab*[2] pour récupérer les récits utilisateurs et le code source. Nous avons créé l'ontologie en utilisant *Pro-*

2. https://docs.gitlab.com/ee/api/rest/index.html

tégé [Mus15] à partir des connaissances de notre partenaire ITK. Pour le traitement du langage naturel, nous utilisons *Gensim* [RS11] avec *word2vec* [MSC+13] et *k-means* [Boc07]. L'analyse formelle de concepts a été réalisée à l'aide de *FCA4J* [GHM22] et l'algorithme de calcul de la base des prémisses propres a été implémenté à l'aide de la méthode décrite par Bazil et al [Baz18]. Nos modèles finaux sont représentés au format *XML* et sont chargés dans *FeatureIDE* [TKB+14].

L'implémentation de notre méthode est disponible en ligne ici :
https://gite.lirmm.fr/tgeorges/Bridging-the-Gap-between-User-Stories-and-Feature-Models-by-Leveraging-Version-Control-Platform/

5.4 Approche détaillée

Nous détaillons dans cette section la synthèse du modèle de caractéristiques et l'illustrons avec un exemple inspiré de notre jeu de données industriel.

Dans notre exemple, les entrées sont les récits utilisateurs des produits, avec le code source associé et un échantillon de l'ontologie de notre partenaire. Le code source est lié aux récits utilisateurs par le biais de fusions (*merges*) dans une instance Gitlab.

5.4.1 Système de contrôle de versions

Pour collecter les récits utilisateurs et le code source associé, nous exploitons les possibilités offertes par le système de contrôle de versions (VCS). Ils sont couramment utilisés dans les projets logiciels et plus particulièrement pour les projets logiciels agiles. Grâce au système de contrôle de versions, nous avons une traçabilité du code source aux récits utilisateurs en utilisant les *commits*. Avec l'API VCS, nous sommes capables de reconstruire le système à partir des validations, des fusions, des changements et de leurs relations.

Les récits utilisateurs des différents produits, présentés dans le tableau 5.1, sont les premières entrées. Le tableau 5.2 représente les fichiers impactés par les fusions qui implémentent les caractéristiques. Dans notre exemple, le code source est représenté par les noms des fichiers impactés par la mise en œuvre des récits utilisateurs. Les relations entre les caractéristiques et le code source sont construites en analysant les fusions liées aux tickets qui traitent les récits utilisateurs grâce à l'API VCS.

Table 5.1 – Récits utilisateurs par produits

	Almond	Vine	Orchard
As a farmer I can refresh the predicted weather		x	x
As a farmer I can CRUD plots	x	x	x
As a farmer with a plot I can edit the parameters of a plot (current season)		x	x
As a farmer with a plot I can sort my plots in the list			x
As a farmer with a plot I can filter my plots in the list			x
As a farmer with a plot I can export observation data for a plot		x	
As a farmer with a plot I know when my plots will be in danger		x	x
As a farmer irrigator I can manage my irrigations and recommendations in my favorite unit	x		
As a farmer irrigator I choose my preferred irrigation unit in my user-settings		x	
As a farmer irrigator I can view my irrigation recommendations in my favorite unit	x	x	
As an admin I can CRUD a farmer		x	x
As an admin I can relaunch all failed simulation	x	x	x

TABLE 5.2 – caractéristiques par fichiers

	admin	weather	farmer	plotList	danger	irrigation	recommendation	export	observation	plot	user-settings	alert
can refresh the predicted weather		x	x									
can CRUD plots			x							x		
can edit the parameters of a plot current season			x							x		
can sort my plots in the list				x						x		
can filter my plots in the list				x						x		
can export observation data for a plot								x	x			
know when my plots will be in danger					x							x
can manage my irrigations and recommendations in my favorite unit						x	x					
choose my preferred irrigation unit in my user-settings						x					x	
can view my irrigation recommendations in my favorite unit						x	x					
can CRUD a farmer	x		x									
can relaunch all failed simulation	x	x										

5.4.2 Décomposition de récits utilisateurs et classifications

Le TALN a deux objectifs dans notre approche. Le premier est la décomposition des récits utilisateurs en séparant le rôle et la caractéristique. Grâce au format spécifique des récits utilisateurs, nous pouvons les analyser pour identifier le premier groupe nominal et la caractéristique comme le contenu qui suit. Le deuxième objectif est le regroupement des caractéristiques pour les organiser en classes (*clusters*).

Décomposition des récits utilisateurs Les techniques de traitement automatique du langage naturel (TALN) permettent de séparer les rôles et les caractéristiques dans les récits utilisateurs. Pour cela, chaque récit utilisateur est traité en utilisant des méthodes de tokenisation, de suppression des mots inutiles (*ou mot vide*), de lemmatisation, puis d'étiquetage des parties du discours (POS-tagging) [BNSM15]. Cette analyse a deux objectifs : séparer le rôle et la caractéristique des récits utilisateurs mais aussi préparer le regroupement.

L'étiquetage nous permet de séparer les rôles de l'ensemble initial de caractéristiques. Par exemple, le récit utilisateur *"As a farmer, I can refresh the predicted weather"* est séparé en : `'Farmer'` (le rôle) et `"refresh the predicted weather"` (la caractéristique). Le rôle est extrait d'un récit utilisateur en considérant le "*premier groupe nominal*". Nous distribuons ensuite les caractéristiques dans différents ensembles, chaque ensemble correspondant à un rôle. Le tableau 5.3 résume la répartition des 12 récits utilisateurs de notre exemple sur les produits et les rôles. Les étapes restantes s'appliquent séparément à chaque ensemble de caractéristiques (un par rôle).

Prenons un récit utilisateur pour détailler le POS-tagging : *"As a Farmer, I can manage my irrigations and recommendations in my favorite unit"*. elle est transformée en : *"manage irrigations recommendations favorite unit"*. L'étiquetage de la partie du discours produit en sortie : "`manage_VB irrigations_NNS recommendations_NNS favorite_JJ unité_NN`", avec "VB" pour verbe, "NN" pour nom, "NNS" pour nom pluriel et "`JJ`" pour adjectif.

Les données résultant de cette étape sont présentées dans le tableau 5.3, dans lequel chaque caractéristique est liée à ses rôles associés. Par exemple, dans le récit utilisateur "*As a Farmer, I can CRUD plots*", le rôle est *"farmer"* et la caractéristique est "*CRUD plots*".

Classification des caractéristiques L'identification et l'association de caractéristiques abstraites à des caractéristiques concrètes ajoutent une dimension de variabilité supplémentaire. La regroupement des caractéristiques concrètes s'effectue grâce à des méthodes de classification.

Ces caractéristiques plus abstraites vont apparaître ultérieurement sous forme de nœuds intermédiaires dans les modèles de caractéristiques générés. Nous partons de l'ensemble de caractéristiques identifiées dans les récits utilisateurs et à ce stade, elles

TABLE 5.3 – Exemple de 12 récits utilisateurs provenant de 3 produits, avec les rôles et les caractéristiques identifiées. Caractéristiques, produits et rôles après le TALN. La ligne *i* correspond au récit utilisateur *i* sans le rôle et la partie *"Je peux", "Je veux pouvoir", "Je sais"*.

		Products			Roles	
Features	No.	Vine	Orchard	Almond	Farmer	Admin
refresh the predicted weather	1	x	x		x	
CRUD plots	2	x	x	x	x	
edit the parameters of a plot (current season)	3	x	x		x	
sort my plots in the list	4		x		x	
filter my plots in the list	5		x		x	
export observation data for a plot	6	x			x	
know when my plots will be in danger	7	x	x		x	
manage my irrigations and recommendations in my favorite unit	8			x	x	
choose my preferred irrigation unit in my user-settings	9	x			x	
view my irrigation recommendations in my favorite unit	10	x		x	x	
CRUD a farmer	11	x	x			x
relaunch all failed simulations	12	x	x	x		x

sont tokenisées, lemmatisées et étiquetées avec du POS-tagging (Part-Of-Speech tagging) pour obtenir un meilleur résultat lors du regroupement. Nous utilisons un modèle Word2vec pré-entraîné [MSC+13] pour la vectorisation, la méthode du coude pour estimer le nombre optimal de clusters et la méthode des k-Means [Boc07] pour le regroupement. Le modèle Word2vec a été pré-entraîné sur le jeu de données *word2vec-google-news-300*, comprenant 3 millions de mots et de phrases. Étant donné que les méthodes de regroupement nécessitent le choix d'un nombre de clusters que nous ne pouvons pas connaître *a priori*, nous avons utilisé la méthode du coude (*Elbow method*) [TWH01] pour identifier le nombre optimal de clusters pour nos données. Cette méthode mesure la distribution de nos données d'entrée et propose une optimisation du nombre de clusters afin de maximiser la cohésion intra-cluster. La méthode du coude

calcule la somme des erreurs quadratiques intra-cluster (*within-cluster-sum of squared Errors*, d'acronyme WSS) et choisit le nombre de clusters correspondant au moment où la métrique WSS cesse de diminuer. La méthode k-Means organise les caractéristiques dans le nombre de clusters donné par la méthode du coude. Avec le regroupement, nous obtenons des groupes de caractéristiques supposées avoir une sémantique cohérente, que nous appelons *caractéristiques abstraites*. La méthode du coude et k-Means ont besoin de données représentées sous forme de nombres. Dans notre cas, nous fournissons à ces méthodes des représentations vectorielles des caractéristiques. Pour ce faire, nous avons utilisé Doc2Vec [LM14] pour *vectoriser* les caractéristiques extraites. La vectorisation projette les récits utilisateurs dans un vecteur de dimension d (d étant le nombre de récits utilisateurs), comme par exemple <0.4125, -1.6098, 0.6047, ..., -1.4257, -1.2321>. Ensuite, l'algorithme k-Means utilise la distance euclidienne pour regrouper les récits utilisateurs les plus proches pour former des clusters. Étant donné un ensemble d'observations $(x_1, x_2, ..., x_n)$, où chaque observation est un vecteur de réels de dimension d, le regroupement k-Means vise à partitionner les n observations en k ($\leq n$) ensembles $S = S_1, S_2, ..., S_k$ de manière à minimiser la somme des carrés intra-cluster. Afin de nommer nos clusters, nous identifions les mots les plus importants en supprimant les mots vides (*stop word*) et en utilisant les mots les plus fréquents restants dans le cluster. Plus précisément, après avoir supprimé les mots vides des caractéristiques regroupées des récits utilisateurs, nous conservons au plus les trois mots importants les plus fréquents pour nommer le cluster correspondant. Le choix des méthodes du coude, k-Means et Doc2Vec est motivé par les bons résultats qu'elles ont obtenus dans des travaux existants [CFM19, AV14], leur documentation complète et l'existence d'outils efficaces. k-Means est une méthode de regroupement couramment utilisée. Dans notre cas, c'était la méthode la plus efficace, comparée à d'autres méthodes que nous avons expérimentées sur notre ensemble de données, y compris LSA/LSI et LDA. Les sujets obtenus pour notre exemple sont présentés dans le tableau 5.4. Ils apparaissent dans la colonne 'AbstractFeatures'. Actuellement, les caractéristiques abstraites n'ont pas de nom généré automatiquement. Pour notre exemple, cela a été fait manuellement. Nous envisageons dans un travail futur de générer ces noms de façon automatisée.

Par exemple, la caractéristique abstraite *"édition des paramètres de la parcelle"* est associée à deux de ses caractéristiques concrètes associées *"pouvoir éditer les paramètres d'une parcelle de la saison actuelle"* et *"pouvoir exporter les données d'observation pour une parcelle"*.

5.4.3 Analyse relationnelle de concepts

Nous nous servons de l'ARC afin d'identifier les relations entre le code source et les spécifications et ainsi construire les modèles de caractéristiques. Le résultat de l'ARC est un ensemble de règles représentant les contraintes logiques entre les objets des contextes relationnels.

Table 5.4 – caractéristiques par caractéristiques abstraites

	refresh predicted weather	crud plots	plot edit parameters	plot unit list	crud farmer
can refresh the predicted weather	x				
can CRUD plots		x			
can edit the parameters of a plot current season			x		
can sort my plots in the list				x	
can filter my plots in the list				x	
can export observation data for a plot			x		
know when my plots will be in danger				x	
can manage my irrigations and recommendations in my favorite unit				x	
choose my preferred irrigation unit in my user-settings				x	
can view my irrigation recommendations in my favorite unit				x	
can CRUD a farmer					x
can relaunch all failed simulation			x		

Les objets sont décrits par un modèle conceptuel comme celui présenté dans la figure 5.6 qui correspond à nos données. Cette figure précise les liens entre les objets tels qu'ils ont été définis dans la figure 5.3.

La figure 5.7 représente le modèle conceptuel implémenté qui est plus proche du modèle physique mis en oeuvre par l'ARC. Il est navigable et les cardinalités entre les récits utilisateurs et les caractéristiques sont un pour un. Une trop grande navigabilité impliquerait une multiplication des cycles et des contraintes générées, c'est pourquoi elle est limitée dans le schéma choisi à ce qui nous a servi pour résoudre notre problème.

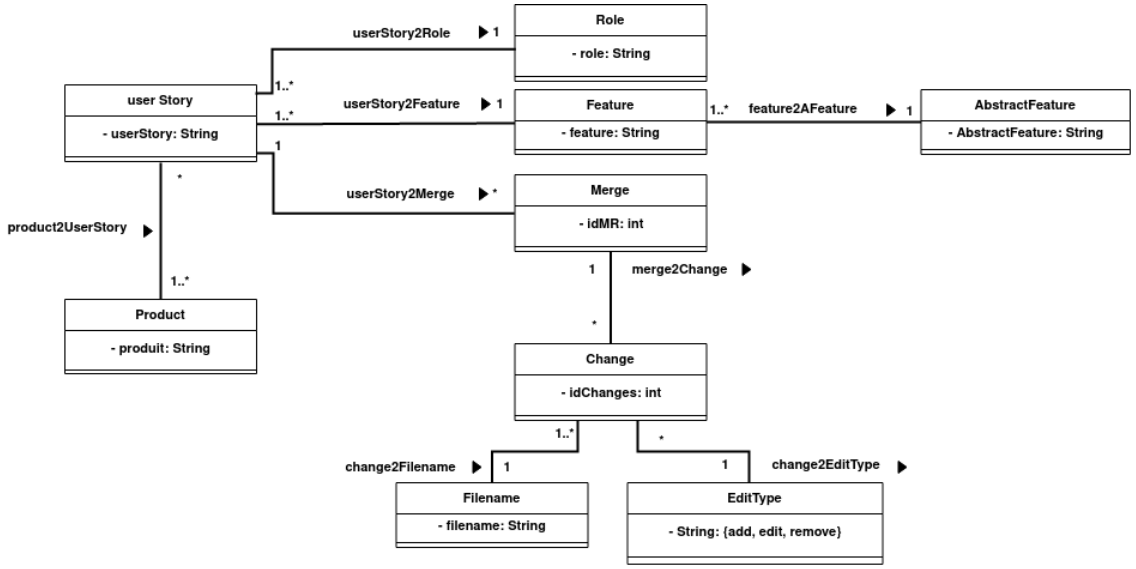


FIGURE 5.6 – Modèle conceptuel de nos données en syntaxe UML. Les classes UML correspondent aux contextes formels et les associations UML aux contextes relationnels.

Les tables du modèle physique tel qu'implémenté en ARC sont présentées dans la figure 5.8. Ce sont les tables représentant les contextes formels et relationnels utilisés par l'ARC.

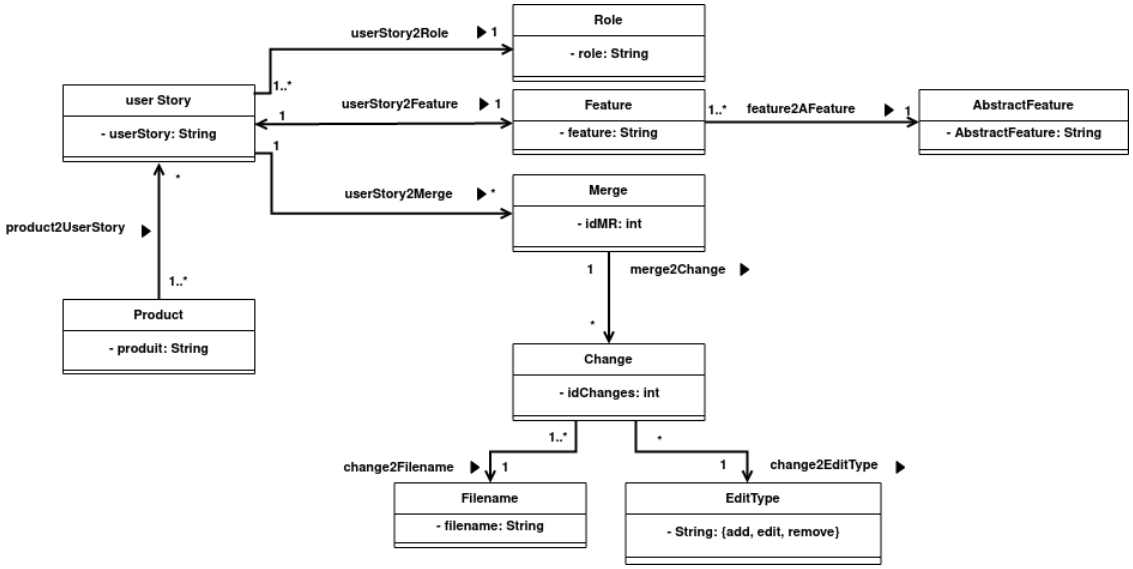


FIGURE 5.7 – Modèle conceptuel implémenté dans l'ARC dans notre cas d'utilisation. Les classes UML correspondent aux contextes formels et les associations UML aux contextes relationnels

L'apport de l'AFC est d'extraire des groupes d'objets en considérant seulement les attributs primitifs, mettant également à jour les relations logiques entre objets ou entre attributs. Dans notre cas, l'AFC nous permet de représenter la variabilité des caractéristiques [LP07b]. Lorsque ces groupes et ces relations logiques concernent les caractéristiques et les caractéristiques abstraites, nous pouvons en tirer des éléments pour la construction du modèle de caractéristiques et de ses contraintes. Dans notre cas, nous exploitons en plus l'ARC pour regrouper les produits, les récits utilisateurs, les caractéristiques, les rôles et le code source en utilisant leur description et leurs relations (par exemple, *un produit a un récit utilisateur*). Nous construisons un contexte formel pour chaque élément d'intérêt :
`product`, `user story`, `role`, `feature`, `abstract feature`, `merge`, `change` et `filename`, ainsi qu'un contexte relationnel pour chaque relation :
`product` vers `user story`, `user story` vers `role`, `user story` vers `feature`, `feature` vers `abstract feature`, `user story` vers `merge`, `merge` vers `change`, `change` vers `filename` et `change` vers `EditType`.

À partir des récits utilisateurs, de leurs caractéristiques abstraites et du code source nous identifions les contraintes logiques. Les contraintes les plus visibles apparaissent lorsque tous les produits ont les mêmes caractéristiques. Dans l'exemple, nos trois produits partagent deux caractéristiques : *« pouvoir créer, lire, mettre à jour et supprimer des parcelles »* et *« pouvoir relancer toutes les simulations échouées »*. Ces deux caractéristiques doivent donc être obligatoires.

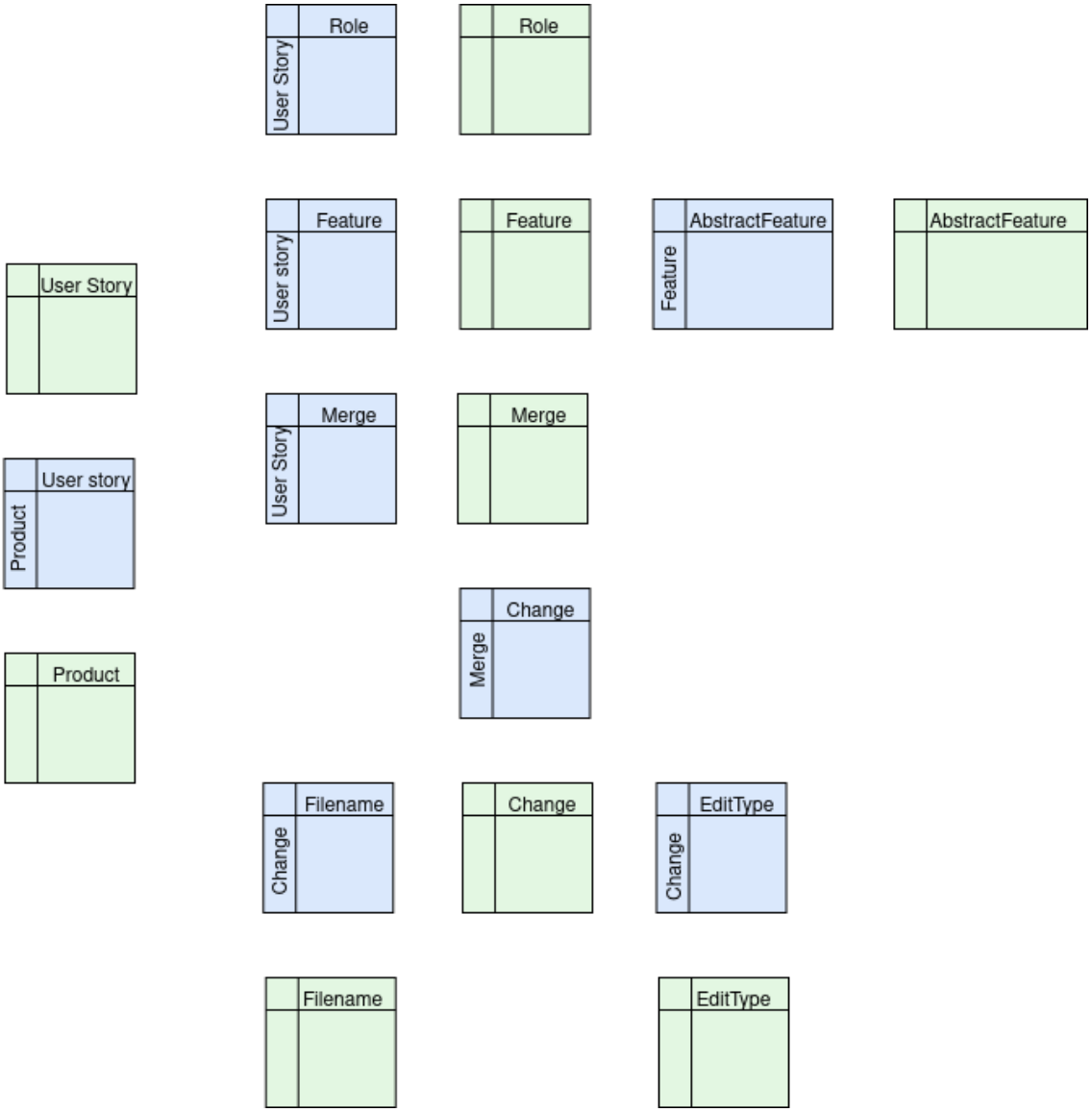


FIGURE 5.8 – Tables telles qu'implémentés dans l'ARC avec les contextes formels en vert et les contextes relationnels en bleu.

À partir d'un contexte formel étendu qui est le contexte formel avec les attributs relationnels, le cadre de l'AFC permet de construire des ensembles d'implications entre les caractéristiques du contexte (attributs primitifs et relationnels). Certains de ces ensembles d'implications, à condition qu'ils soient non redondants et suffisants pour inférer toute autre implication détenue par les objets, sont appelés bases d'implications. Deux de ces bases ont été considérées dans ce travail : la *base de Duquenne-Guigues* [GD86, GMNR05] et la *base d'implications à prémisse propre* [RPK11].

Ces bases ont différentes propriétés : la base de Duquenne-Guigues est de cardinalité minimale et la base d'implications à prémisse propre est composée de règles dont les prémisses sont minimales et dont la conclusion contient une seule caractéristique.

Une règle est un moyen d'établir des liens logiques entre les attributs d'un contexte formel. Une règle est composée d'un ensemble de prémisses, d'un symbole d'implication et d'un ensemble de conclusions comme montré ci-dessous :

$$premisses \Rightarrow conclusions$$

Par exemple

« peut relancer toutes les simulations échouées » et « peut CRUD [3] des parcelles »
$\Rightarrow$ « parcelle unité liste ».

L'implication précédente est une réécriture à destination de FeatureIDE d'une implication dont la forme calculée est :

exist_products2UserStories(exist_userStories2features(peut relancer toutes les simulations échouées)), exist_products2UserStories(exist_userStories2features(peut CRUD des parcelles)) => exist_products2UserStories(exist_userStories2features(exist_features2AbstracFeatures(parcelle unité liste)))

En pratique dans sa forme calculée, l'implication a pour prémisse deux attributs relationnels ciblant des concepts introduisant des caractéristiques concrètes et pour conclusion un attribut relationnel ciblant un concept introduisant une caractéristique abstraite. Cette règle a été calculée sur le contexte des récits utilisateurs. Pour le dire plus simplement, cette règle établit un lien entre deux caractéristiques concrètes et une caractéristique abstraite. Elle signifie que si une configuration cible les deux premières caractéristiques, alors elle cible également le cluster (caractéristique abstraite) en conclusion. Si les caractéristiques « peut relancer toutes les simulations échouées » et « peut CRUD des parcelles » sont présentes alors le cluster comprenant les termes « parcelle unité liste » doivent l'être aussi.

De manière générale, si tous les attributs relationnels ou primitifs dans les prémisses sont présents dans une configuration, alors tous les attributs relationnels ou primitifs dans la conclusion doivent également être présents dans cette configuration. Grâce à l'ARC, les groupes formés sur une catégorie d'objets, par exemple sur les fichiers se propagent en remontant le long des relations comme montré par la figure 5.8, par exemple jusqu'aux fusions, puis jusqu'aux récits utilisateurs. Nous obtenons une liste de règles représentant les relations logiques entre les attributs, ceux-ci impliquant des concepts

3. `Create`, `Read`, `Updtate`, `Delete`

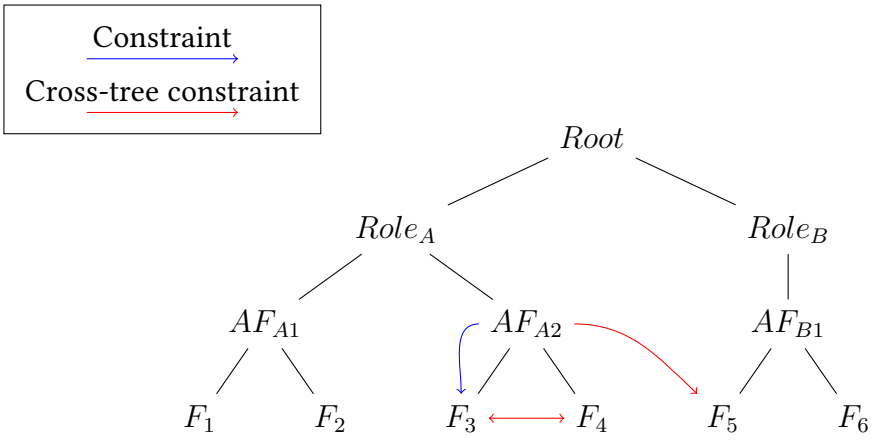


FIGURE 5.9 – Modèle rôle caractéristique illustrant les contraintes

regroupant l'un des éléments suivants : les produits, les récits utilisateurs, les rôles, les caractéristiques, les caractéristiques abstraites, les fusions, les modifications ou les noms de fichiers.

Chaque règle a un support absolu [4], généré par l'ARC et correspondant au nombre d'objets possédant les prémisses de la règle.

$$< support > premises \Rightarrow conclusions$$

Le niveau 0 exprime qu'aucun objet ne dispose de toutes les prémisses de la règle, ce qui peut être interprété comme une incompatibilité entre les concepts dans les prémisses. Un niveau strictement supérieur à 0 signifie que plusieurs objets disposent de toutes les prémisses de la règle, et si le niveau est égal au nombre d'objets, cela signifie que tous les objets disposent de toutes les prémisses de la règle. Nous interprétons le support n d'une règle construite sur le contexte formel des produits pour extraire les contraintes correspondantes pour nos modèles de caractéristiques. Une *contrainte* est le plus souvent utilisée pour décrire une relation binaire entre deux attributs issus de deux concepts liés (ancêtre ou descendant). Une contrainte qui ne correspond pas à ce cas est appelée une *contrainte cross-tree*. Ainsi, une contrainte est construite à partir d'une règle et correspond à l'une des situations suivantes :

— Une relation entre des clusters (caractéristiques abstraites) et des caractéristiques concrètes. La relation peut être cross-tree, si ce n'est pas une arête dans l'arbre de caractéristiques.

4. Le support est parfois le nom donné au nombre d'objets possédant les prémisses divisé par le nombre d'objets total. Ici nous utiliserons le terme support pour parler du support absolu.

— Une relation entre des caractéristiques concrètes (uniquement cross-tree).

De telles contraintes sont illustrées dans la figure 5.9. Dans cet exemple, la contrainte $AF_{A2} \Rightarrow F_5$ est une contrainte cross-tree qui implique que toutes les configurations contenant AF_{A2} doivent également contenir F_5. Au contraire, la contrainte $F_1 \Rightarrow AF_{A1}$ est une contrainte de raffinement, représentée par une arête de l'arbre, ce qui signifie que lorsque la caractéristique F_1 est incluse dans une configuration, sa caractéristique mère abstraite AF_{A1} doit également être incluse dans cette configuration. Enfin, la contrainte $AF_{A2} \Rightarrow F_3$ est une contrainte additionnelle impliquant que si la caractéristique abstraite AF_{A2} fait partie d'une configuration, alors la caractéristique F_3 doit en faire également partie.

La transition des règles aux contraintes est détaillée ci-dessous selon le support n de la règle :

Si $n = 0$: le niveau zéro signifie qu'aucun produit ne dispose de toutes les prémisses (caractéristiques). Autrement dit, dans les prémisses, au moins un élément ne peut pas être présent en même temps que tous les autres. À partir de la règle : *<0>* $A \wedge B \wedge C \implies D$ nous créons la contrainte cross-tree $\neg$ *(* $A \wedge B \wedge C$*). A, B, C et D* sont des attributs des objets de notre modèle impliquant des récits utilisateurs, des caractéristiques, des rôles ...

Si $n > 0$: certains produits ne disposent pas de toutes les prémisses de la règle. Pour déduire les contraintes dans ce cas, nous devons interpréter le résultat pour le treillis. Cela peut signifier qu'une contrainte avec un groupe *ou* ou un groupe *alternatif* est nécessaire.

Si $n = nombredeproduits$: tous les produits disposent de toutes les prémisses de la règle. Les attributs et caractéristiques des objets dans les prémisses impliquent les caractéristiques dans la conclusion. Nous utilisons cette règle pour construire les relations obligatoires du modèle de caractéristiques.

Des contraintes particulières sont identifiées par des règles de niveau 0, par exemple avec le cluster *« liste des parcelles »* et les caractéristiques *« pouvoir créer, lire, mettre à jour et supprimer des parcelles »* et *« pouvoir relancer toutes les simulations échouées »*. Ce cluster et ces caractéristiques ne peuvent pas être présents en même temps.

En plus des règles, de manière générale, le treillis de concepts peut être utilisé pour visualiser les groupes de concepts et pour trouver les groupes *ou* et *alternatifs* dans le modèle de caractéristiques et le modèle de rôles-caractéristiques.

5.4.4 Synthèse du modèle de caractéristiques et du modèle de rôles-caractéristiques

Le modèle de caractéristiques synthétisé comprend deux types de caractéristiques : celles identifiées à partir des récits utilisateurs et celles abstraites identifiées à partir du regroupement (*clustering*). Le modèle de caractéristiques synthétisé pour notre exemple est présenté dans la figure 5.10. Dans notre exemple, trois contraintes ont été déduites. Les deux premières sont des caractéristiques obligatoires : *« pouvoir créer, lire, mettre à jour et supprimer des parcelles »* et *« pouvoir relancer toutes les simulations échouées »*. La troisième est une contrainte entre différentes parties de l'arbre (*cross-tree constraint*) qui évite que le cluster *« liste des parcelles »* et les caractéristiques *« pouvoir créer, lire, mettre à jour et supprimer des parcelles »* et *« pouvoir relancer toutes les simulations échouées »* soient choisis en même temps.

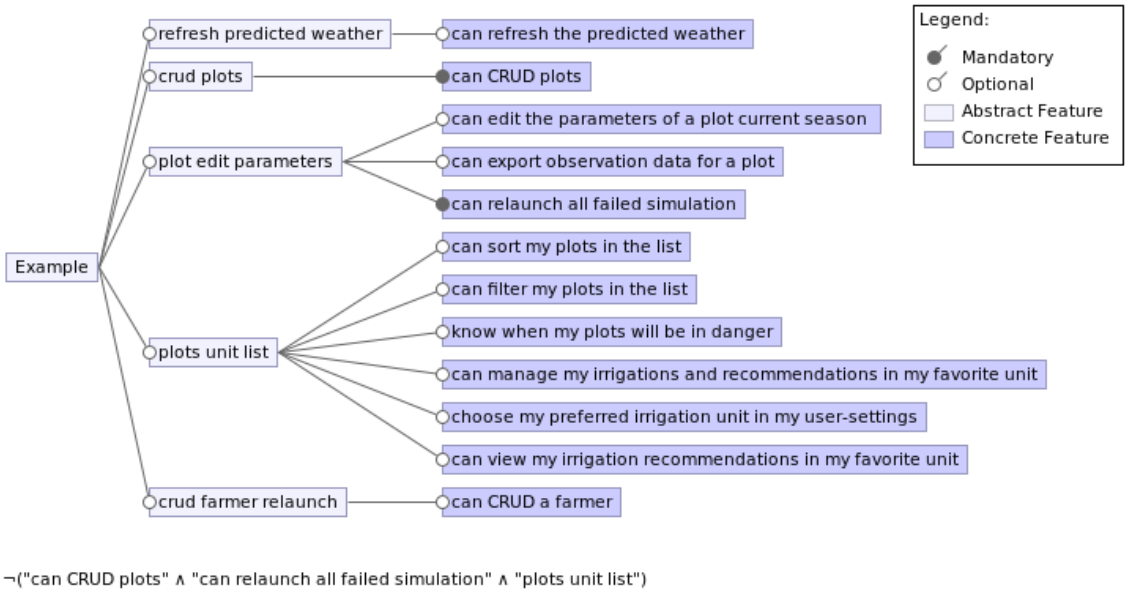


FIGURE 5.10 – Exemple de modèle de caractéristiques

Les modèles de rôles-caractéristiques ont un niveau supplémentaire. Ce ne sont pas exactement des modèles de caractéristiques car le rôle n'est pas une vue fonctionnelle. Le modèle de rôles-caractéristiques synthétisé pour notre exemple est présenté dans la figure 5.11. Les rôles sont « agriculteur » et « administrateur ». Dans notre modèle de rôles-caractéristiques, les deux rôles sont obligatoires car chaque produit possède un rôle agriculteur et un rôle administrateur. Des exemples de récits utilisateurs incluent *« En tant qu'agriculteur, je peux créer, lire, mettre à jour et supprimer des parcelles »* et *« En tant qu'administrateur, je peux relancer toutes les simulations échouées »*.

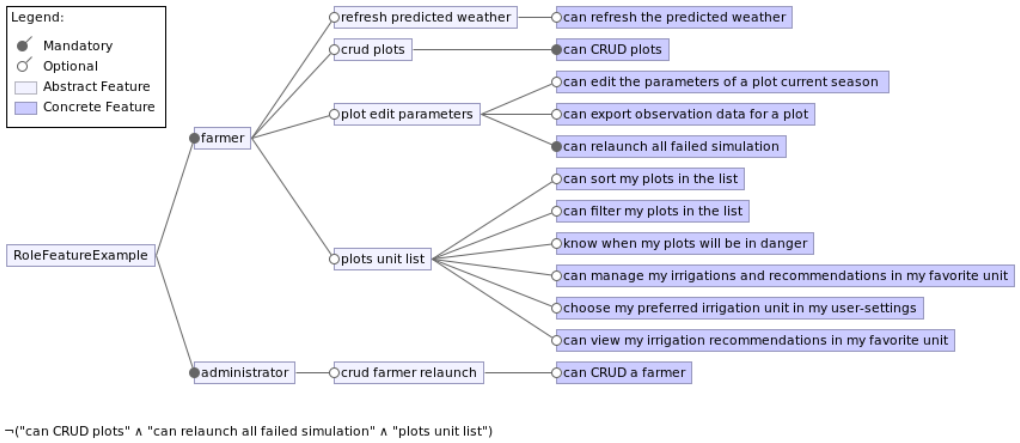


FIGURE 5.11 – Exemple de modèle de rôles-caractéristiques

5.5 Etude de cas

Afin d'évaluer le processus proposé, nous avons mené une étude de cas avec notre partenaire industriel. Nous avons défini un protocole pour comparer les modèles de caractéristiques synthétisés avec notre processus avec ceux créés manuellement par notre partenaire industriel. Actuellement, notre partenaire industriel n'a pas encore pleinement adopté l'ingénierie des LPL. Par conséquent, nous n'avons pas de modèles de caractéristiques concrets définis manuellement par l'équipe informatique.

Nous avons organisé un atelier (voir Figure 5.12) où un expert de l'équipe de développement a été invité à construire son propre modèle de caractéristiques à partir d'un ensemble de caractéristiques. Ensuite, pour la comparaison, nous avons confronté les deux modèles de caractéristiques en interagissant avec cet expert. Plus le modèle de caractéristiques synthétisé se rapproche de celui de l'expert, plus il est précis. Lorsqu'un modèle de caractéristiques est précis, il devient utile pour l'équipe informatique.

Les caractéristiques concrètes et abstraites ont été identifiées par la méthode détaillée dans les sections précédentes de ce chapitre, avec du TALN sur des récits utilisateurs et la classification des caractéristiques.

Nous avons sollicité l'expertise d'un développeur en ITK qui compte 8 années d'expérience, dont 5 années spécifiquement consacrées à la plateforme en question. Il a participé à l'implémentation de nombreuses caractéristiques présentées lors de l'atelier, mais il a également été impliqué dans leur développement (relecture de code, travail en duo, écriture des spécifications, etc.).

Il est important de noter que cet expert a fait partie des personnes interviewées lors de notre étude sur la perception de la migration.

Pour l'étude de cas, nous avons utilisé les récits utilisateur, le code source et l'ontologie de l'instance Gitlab de notre partenaire industriel. L'ensemble de données comprend

FIGURE 5.12 – Atelier organisé avec l'expert

127 récits utilisateur, 6 produits, 60 fusions (merges) et des milliers de fichiers. Nous avons identifié 15 rôles et regroupé 127 caractéristiques en 42 clusters. Pour la validation avec l'expert, nous avons pris un échantillon de 60 caractéristiques dans 7 clusters, afin de rendre la validation réalisable par des humains. Ce modèle de caractéristiques est présenté dans la figure 5.13.

L'objectif de la validation avec l'expert est de déterminer la précision, la cohérence et l'intérêt d'un tel processus automatisé.

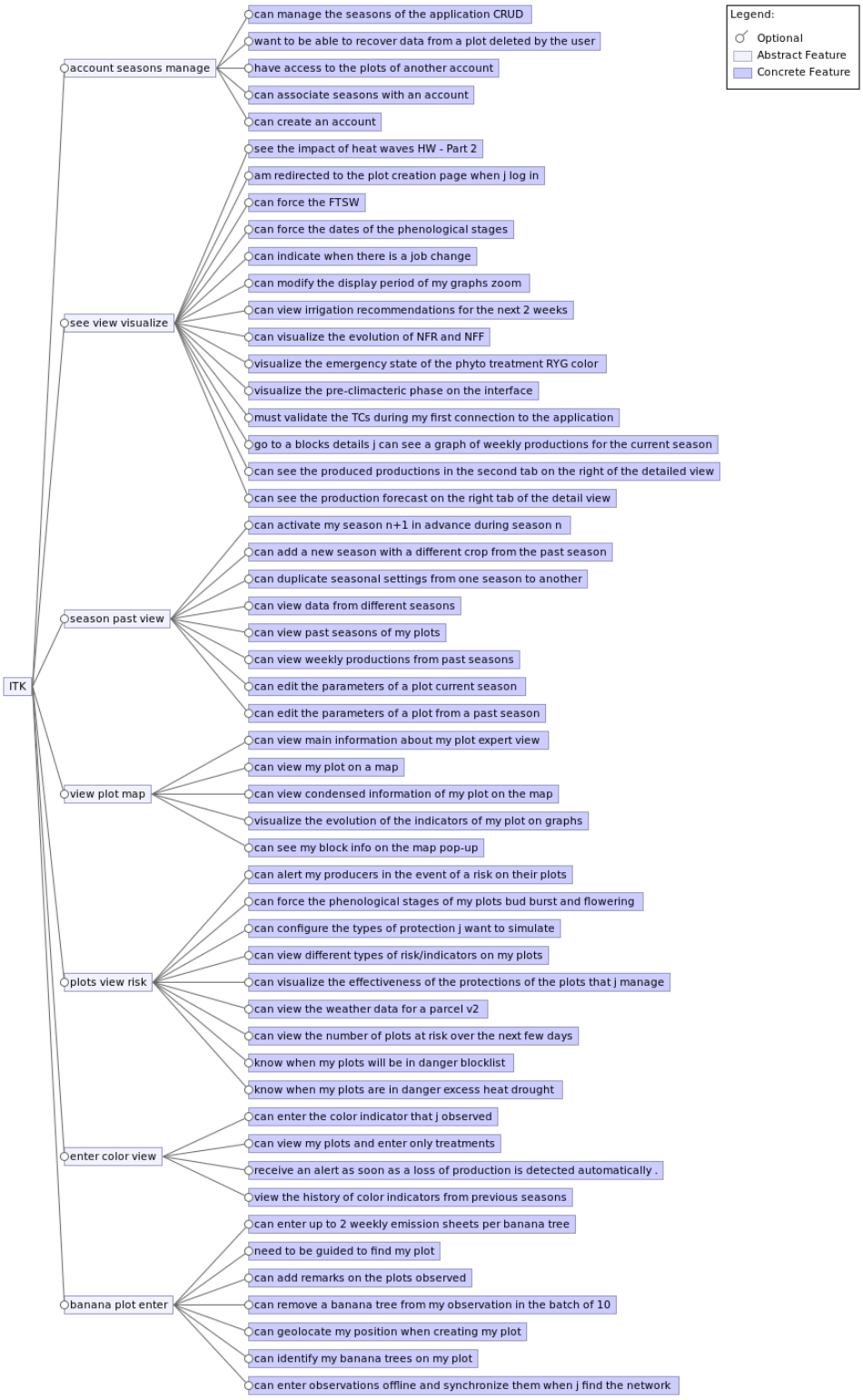


FIGURE 5.13 – modèle de caractéristiques utilisé dans l'évaluation

5.5.1 Protocole

Tout d'abord, une introduction et une description des modèles de caractéristiques et des LPL ont été présentées à l'expert. Ensuite, nous avons suivi la procédure décrite ci-dessous :

- *Étape (1)*, nous avons demandé à l'expert de construire *son* modèle de caractéristiques avec un ensemble de caractéristiques ;
- *Étape (2)*, nous avons présenté à l'expert *notre* modèle de caractéristiques et nous lui avons demandé de le réorganiser si nécessaire ;
- *Étape (3)*, nous avons demandé à l'expert de réorganiser *son* modèle de caractéristiques, si nécessaire.

Dans notre validation, nous avons évité tout biais introduit par notre présence lors de l'atelier, en laissant d'abord notre interlocuteur construire le modèle de caractéristiques sans aide ni indications.

- (1) Nous donnons à l'expert les caractéristiques et les caractéristiques abstraites, et nous lui demandons de produire un modèle de caractéristiques ;
- (2) Nous donnons à l'expert notre modèle de caractéristiques et nous lui demandons de le réorganiser ;
- (3) Nous redonnons à l'expert le modèle de caractéristiques qu'il a construit la première fois et nous lui demandons s'il souhaite le modifier.

À partir du modèle de caractéristiques créé par l'expert, il y avait trois résultats possibles selon la similarité entre ses résultats et les nôtres :

- *Cas 1 :* le modèle de caractéristiques de l'expert présente uniquement des clusters *similaires* à ceux de notre modèle de caractéristiques, cela peut signifier que notre modèle de caractéristiques est pertinent pour l'entreprise.
- *Cas 2 :* le modèle de caractéristiques de l'expert présente des clusters *similaires* à ceux de notre modèle de caractéristiques et il combine des caractéristiques de différents clusters, donc notre modèle de caractéristiques représente partiellement le système.
- *Cas 3 :* le modèle de caractéristiques de l'expert n'a pas de clusters *similaires* à ceux de notre modèle de caractéristiques. L'expert réorganise les caractéristiques de différents clusters. Notre modèle de caractéristiques doit donc utiliser plus de connaissances de l'ontologie pour se rapprocher du domaine.

À chaque étape, nous avons engagé une discussion sur le processus, au cours de laquelle nous avons sollicité l'expert pour évaluer son niveau de confiance dans le modèle de caractéristiques qu'il avait développé, en utilisant une échelle de notation de 1 à 5. Un score de 1 indiquait qu'il avait peu de confiance dans le modèle, tandis qu'un score

de 5 signifiait une confiance totale. Cette discussion a également permis de déterminer la similarité des clusters entre ceux de l'expert et les nôtres.

Nous avons également chronométré la durée nécessaire pour chaque étape. La première étape, qui consistait à trier et à catégoriser les caractéristiques manuellement, s'est avérée être la plus chronophage. En revanche, les étapes de correction du modèle automatisé puis du modèle manuel étaient moins longues. Cette différence s'explique par la meilleure compréhension des caractéristiques acquises au cours de la première étape, ainsi que par la réflexion préliminaire effectuée.

Nous avons sollicité l'expert pour évaluer la difficulté de chaque étape sur une échelle de 1 à 5. L'expert a indiqué que la première étape était la plus complexe, tandis que les étapes suivantes étaient plus simples à ses yeux.

Les mesures sont présentées dans le tableau 5.5 pour l'exemple et dans le tableau 5.6 pour le cas réel.

Table 5.5 – Métriques de l'exemple jouet

Exemple	Temps minutes	Difficulté (1-5)	NbClusters	Confiance (1-5)
À la Main	10	3	6	3
Automatisé	5	2	5	3
À la Main	5	2	6	4

Table 5.6 – Métriques de l'exemple réel

Réel	Temps minutes	Difficulté (1-5)	NbClusters	Confiance (1-5)
À la main	30	4	8	3
Automatisé	10	2	7	3
À la main	15	2	7	4

Enfin, nous avons consigné les remarques et les commentaires de l'expert pour chaque étape. L'expert a souligné que la première étape était la plus exigeante en termes de temps et de complexité. Elle lui a permis de se familiariser avec les caractéristiques, mais elle était également fastidieuse. En revanche, la deuxième étape a apporté un nouvel éclairage sur les caractéristiques et a permis d'introduire une perspective organisationnelle. Cela a enrichi la dernière étape, qui consistait à corriger le modèle initial créé manuellement, en améliorant à la fois les catégories et le niveau de confiance.

5.5.2 Analyse

L'expert connaît les récits utilisateurs et n'a fait aucun commentaire sur les rôles ou les caractéristiques identifiés à partir de ces récits utilisateurs. Cela confirme que les

premières étapes de notre processus n'ont eu aucun impact négatif lors du traitement de l'ensemble de données initial.

La première étape a été la plus longue, l'expert a trouvé difficile la création des premiers clusters (Figure 5.12). La confiance de l'expert dans les modèles de caractéristiques lors de la première étape était de $3/5$. Dans la deuxième étape, lors de la comparaison avec nos modèles de caractéristiques synthétisés, la note de l'expert est de $3/5$. L'expert a noté des différences fonctionnelles avec son travail. En particulier, il a fait l'observation que notre modèle de caractéristiques est plus proche de l'implémentation et que le modèle de caractéristiques construit manuellement est plus abstrait.

Dans la troisième étape, l'expert a édité son modèle de caractéristiques (voir Figure 5.14) et a donné une note de confiance de $4/5$. Cela signifie que notre modèle de caractéristiques synthétisé, même s'il ne lui a pas fourni une solution complète, a aidé l'expert à affiner son modèle et à obtenir une vue plus précise sur l'ensemble du système. L'expert a déclaré que : *« le modèle de caractéristiques synthétisé permet de gagner du temps et c'est une manière plus facile de corriger son modèle de caractéristiques que de le construire manuellement depuis zéro »*. L'expert n'a pas trouvé d'incohérence dans le modèle de caractéristiques synthétisé. L'expert a dit que : *les contraintes transversales identifiées sont trop nombreuses, avec beaucoup de faux positifs*". Dans la troisième étape, l'expert a remarqué certaines erreurs dans son modèle qui devaient être corrigées. Par exemple, l'expert a affiné un cluster en le divisant en deux clusters distincts. Le cluster "manage my plots" a été divisé en "plot parameters edit" et "plot list". À la fin, le modèle de caractéristiques comprenait des clusters plus spécifiques, correspondant à des points fonctionnels plus ciblés. L'expert a également renommé certains clusters en utilisant des noms donnés par notre modèle.

Sur l'exemple, l'expert a construit 4 clusters à l'*Étape 1* et 5 clusters à l'*Étape 3*. Lors de l'Étape 2, l'expert a suggéré d'ajouter deux clusters et d'en fusionner un dans notre modèle de caractéristiques. Sur l'échantillon plus large, l'expert a créé 10 clusters à l'*Étape 1*. L'expert a ajouté 2 clusters, portant le total à 11 clusters dans notre modèle lors de l'*Étape 2*. Ensuite, à l'*Étape 3*, l'expert a renommé les clusters et réorganisé les caractéristiques.

Dans cette étude de cas, nous avons évalué notre synthèse de modèles de caractéristiques avec un expert de notre partenaire industriel. Notre objectif était de comparer les modèles de caractéristiques générés par notre approche automatisé avec ceux créés manuellement par l'expert. L'atelier a montré que notre processus automatisé pouvait produire des modèles de caractéristiques similaires à ceux créés manuellement par l'expert. L'expert a trouvé que la première étape de création initiale des clusters de caractéristiques était la plus exigeante, mais que les étapes suivantes de réorganisation et de correction étaient facilitées par notre modèle de caractéristiques. L'expert a noté que notre processus permettait de gagner du temps et de faciliter la correction du modèle de caractéristiques par rapport à une construction manuelle complète. En résumé, cette étude de cas a montré que notre processus de synthèse de modèles de caractéristiques

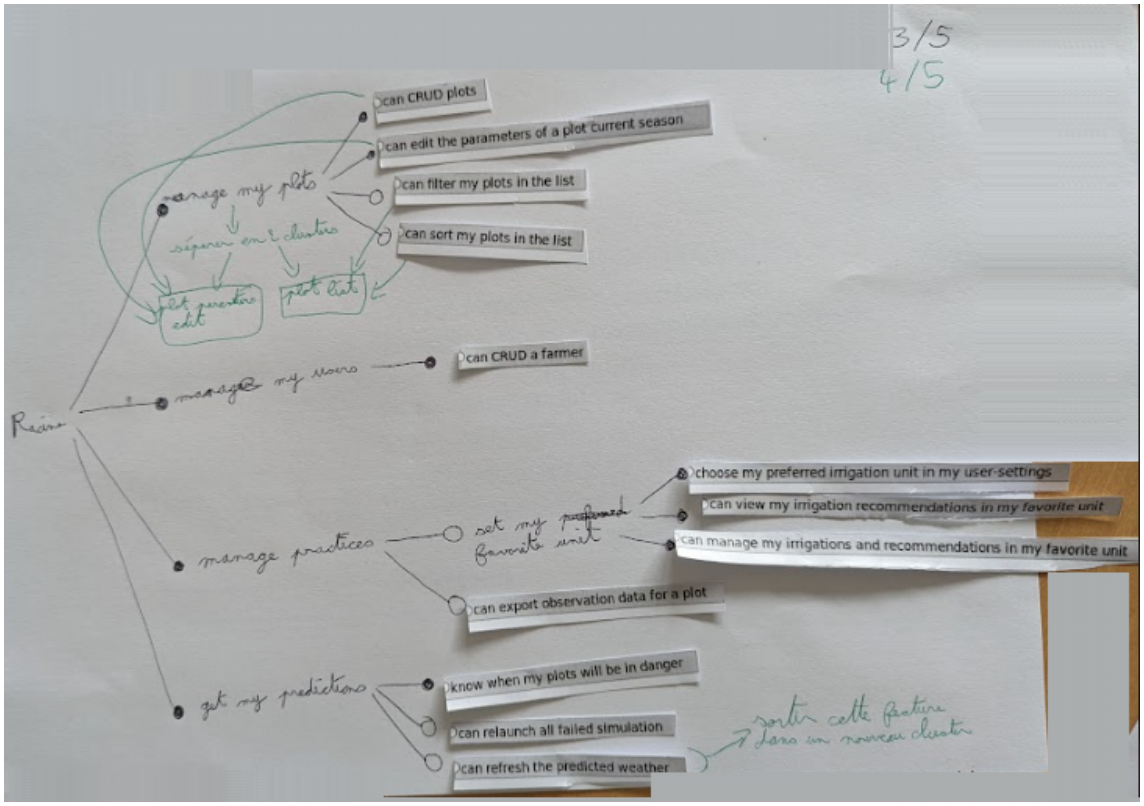


FIGURE 5.14 – Résultats de l'atelier sur l'exemple

peut être un outil utile, en particulier lorsque des experts du domaine sont disponibles pour affiner le modèle. Il permet d'obtenir des modèles de caractéristiques plus proches de l'implémentation, ce qui peut être bénéfique dans le contexte de l'ingénierie logicielle.

5.5.3 Discussion

Principales conclusions

Nous avons présenté dans ce chapitre un processus automatisé pour synthétiser des modèles de caractéristiques à partir de produits logiciels existants construits dans le cadre de projets agiles. En utilisant le traitement du langage naturel, la vectorisation, le regroupement (clustering), l'intégration d'ontologies, ainsi que l'analyse conceptuelle formelle et relationnelle, nous avons pu identifier et affiner les caractéristiques concrètes et abstraites, ainsi qu'établir des relations de traçabilité entre les récits utilisateurs et le code source. Notre processus repose sur le traitement du langage naturel pour identifier les rôles et les caractéristiques. Grâce à ce traitement, nous regroupons également les caractéristiques en caractéristiques abstraites. Ensuite, nous utilisons l'analyse relation-

nelle de concepts en exploitant les relations entre nos concepts. Cette approche offre une manière complète d'extraire des modèles de caractéristiques pertinents à partir de produits existants et peut être facilement modifiée par des experts pour produire le modèle de caractéristiques réel de la ligne de produit.

Menaces pour la validité

Une menace concerne la généralisation de l'approche. En effet, l'étude de cas a été réalisée avec notre partenaire industriel, et l'outil a été conçu en utilisant la technologie utilisée par ce partenaire. En particulier, les récits utilisateurs sont décrits au sein de GitLab. Cependant, nous nous attendons à ce que notre approche soit applicable à un large éventail de projets. La définition des récits utilisateurs comme exigences et leur gestion conjointe avec le code source via des fusions dans un système de contrôle de versions sont une pratique courante dans les projets logiciels actuels. Cette façon de travailler est populaire dans les entreprises qui utilisent l'agilité dans leurs projets logiciels. Notre approche est donc applicable à un ensemble de projets utilisant les récits utilisateurs pour les décrire et utilisant des fusions gérées par un système de contrôle de versions pour implémenter leurs caractéristiques. Même avec une mise en œuvre technique différente d'un tel processus, par exemple avec des récits utilisateurs externalisés du système de contrôle de versions, l'approche reste applicable, avec des outils différents.

La validation de notre approche a été réalisée sur des produits existants de notre partenaire industriel. Lors de l'atelier avec l'expert, nous avons comparé deux manières de produire le modèle de caractéristiques : ex nihilo ou en adaptant un modèle de caractéristiques généré grâce à notre approche. À partir de ce protocole, nous concluons que notre modèle de caractéristiques généré est pertinent et utile. Cependant, une menace pour la validité réside dans le fait que nous avons fourni à l'expert un seul modèle de caractéristiques généré automatiquement en utilisant notre approche. Ainsi, d'une part, il se peut que toute information générée à partir des récits utilisateurs, sans utiliser notre approche, puisse être d'une grande aide pour construire le modèle de caractéristiques. D'autre part, nous n'avons pas évalué indépendamment les avantages de chaque partie de notre processus. Nous prévoyons donc d'organiser un deuxième atelier avec des experts afin de leur fournir différents artefacts générés, et de comparer la pertinence des différentes parties de notre approche.

Un autre biais dans le processus de validation est lié au fait que les retours proviennent d'un seul expert. Une validation plus robuste, impliquant un plus grand nombre d'experts, est nécessaire pour atténuer ce biais. C'est l'une des perspectives de ce travail. Nous avons cependant choisi un expert qui a de l'expérience dans le développement de la plate-forme. Ses connaissances du domaine métier et des récits utilisateurs nous ont confortées dans notre choix de mener cet atelier avec un seul expert.

Pour une analyse plus approfondie, nous avons envisagé de calculer plusieurs métriques, telles que la distance d'édition d'arbres nécessaire pour ajouter ou déplacer un

nombre donné de caractéristiques entre l'un des modèles et l'autre. Cela nous permettrait de mettre en évidence l'écart entre ce que l'expert a réalisé et ce que notre processus a produit. Cette approche nous permettrait de mieux quantifier nos résultats auprès de nouveaux experts.

5.5.4 Perspective

Cette méthode s'inscrit dans le processus d'identification et de localisation des caractéristiques. La validation de notre approche est conçue pour évoluer au fil de futurs ateliers. Actuellement, cette validation repose sur un atelier unique impliquant un expert. Bien que cela puisse ne pas être totalement représentatif, cela fournit néanmoins des indicateurs préliminaires quant aux résultats que l'on peut attendre. Il est essentiel de noter que nous n'avons pas relevé d'incohérence dans les résultats obtenus, ce qui renforce la confiance dans l'approche utilisée. En se basant sur le témoignage de l'expert, on peut affirmer que les résultats obtenus sont en conformité avec les attentes et qu'ils peuvent, dans leur état actuel, au moins représenter le système de manière adéquate. Aujourd'hui ces méthodes et approches pourraient être améliorées en prenant en compte de récentes études [Sch23] sur l'usage des méthodes Elbow et k-means. Elles précisent les paramètres optimaux et les alternatives efficaces à l'utilisation de Elbow avec k-means.

Les liens avec le code servent à enrichir les règles. Dans un second temps, elles pourraient permettre également de localiser précisément le code correspondant aux caractéristiques. Nous avons déjà identifié les fichiers associés et avons expérimenté l'utilisation des arbres syntaxiques abstraits (AST) pour localiser la caractéristique dans le code. Une finalité de ces travaux serait de valoriser nos résultats afin de développer une méthodologie allant du récit utilisateur jusqu'au modèle de caractéristiques, puis au code proprement dit.

5.6 Travaux connexes

Principes Agiles des Lignes de Produits Logiciels

Il est pertinent de penser à combiner les approches Agiles et l'ingénierie des lignes de produits logiciels, car les deux ont la réutilisabilité comme objectif. Cependant historiquement les LPL sont des processus lourds et rigides. Une revue systématique de la littérature sur l'ingénierie Agile de lignes de produits logiciels a été décrite dans [DPAG11]. Da Silva et al. ont cartographié les études sur les lignes de produits logiciels Agiles [dSdMSNO$^+$11]. Hanssen a identifié des facteurs favorisant pour l'ingénierie Agile des lignes de produits logiciels [Han11]. Il a mis en évidence les natures différentes des deux approches, la LPL étant un développement « proactif et planifié », tandis que le développement Agile favorise un développement « réactif et axé sur le changement ». Parmi

les facteurs favorables, on peut trouver ce conseil : maintenir une continuité entre le développement de produits et les actifs principaux ; et simplifier la gestion des exigences. En proposant un processus de réingénierie qui prend en compte une chaîne d'artefacts connectés (des récits utilisateurs aux noms de fichiers du code source), notre approche vise à assurer une partie de cette continuité. Fonder le processus sur les récits utilisateurs contribue à maintenir les exigences dans l'esprit du développement Agile.

Systèmes de Contrôle de Versions et Lignes de Produits Logiciels

Les systèmes de contrôle de versions sont une pratique courante pour gérer les versions d'un logiciel, mais ils gèrent difficilement les variantes. Les auteurs de [LSBG21] passent en revue les stratégies existantes qui ont été proposées dans les *systèmes de contrôle de variation*. Une proposition récente est ECSEST (Extraction and Composition for Systems Evolving in Space and Time), qui vise à prendre en charge la composition de variants tout en gérant les révisions de caractéristiques [MOA+22]. Dans notre travail, nous n'abordons pas spécifiquement le problème de la maintenance des révisions de différentes variantes et des informations sur l'emplacement des caractéristiques. Notre objectif est d'exploiter la connexion entre les récits utilisateurs et les modifications du code pour identifier et maintenir un modèle de variabilité. Une méthode pour la coévolution des caractéristiques et du code a été proposée avec l'outil FEVER [DvDP18]. Il repose sur les commits dans un système de contrôle de versions et maintient une correspondance entre les caractéristiques et les éléments. La proposition est appliquée au modèle de variabilité du noyau Linux, qui est exprimé dans le langage Kconfig [SB22]. Une différence avec notre travail est que nous utilisons la chaîne de correspondances qui va des récits utilisateurs aux fichiers pour synthétiser un modèle de caractéristiques. Nous utilisons également des techniques de traitement du langage naturel pour identifier les caractéristiques et l'analyse relationnelle de concepts pour identifier les contraintes logiques.

Exigences et spécification

Les travaux de Classen et al. [CHS08] proposent une extension de la définition de caractéristique axée sur la résolution de problèmes. Ce travail peut nous permettre d'enrichir nos définitions et de mettre en perspective les récits utilisateurs en tant que caractéristiques. Ces travaux nous aident à mieux situer nos récits utilisateurs, car chacun d'entre eux représente un ajout avec un objectif clairement défini. Une approche similaire à la nôtre [SLB+11] utilise les exigences pour construire des modèles de caractéristiques. Dans leur approche, les exigences sont formatées et les contraintes sont explicitées, contrairement à notre approche où nous exploitons l'AFC et l'ARC pour identifier les contraintes entre les caractéristiques. Li et al. [LSSF20] ont proposé une approche automatique similaire à la nôtre pour l'identification de caractéristiques, en particulier

en utilisant Doc2Vec et une méthode de clustering. Ils analysent la connaissance des experts en tant qu'exigence dans le contexte d'une entreprise. Leur approche est associée à une interface graphique permettant la correction des caractéristiques la où dans notre approche nous utilisons une ontologie. Une étude des principes d'identification des caractéristiques a été menée par Nešić et al. [NKSB19]. Cette étude explore les différentes techniques utilisées, répertorie les études traitant de l'identification des caractéristiques et valide leurs résultats à l'aide d'entretiens. Ils ont regroupé ces principes en huit phases et les ont classés par pertinence décroissante pour leur application lors de la construction de modèles de caractéristiques. Les premières phases englobent les principes pertinents pour les activités préalables à la modélisation, telles que la planification ou l'identification des sources d'informations. Les phases suivantes concernent les principes pertinents pour la modélisation effective et la validation du modèle de caractéristiques obtenu.

Traitement du Langage Naturel appliqué aux exigences dans les Lignes de Produits Logiciels

Bakar et al. ont réalisé une revue systématique de la littérature sur les approches qui extraient des caractéristiques à partir des exigences en langage naturel dans les lignes de produits logiciels [BKS15]. Niu et al. traitent des documents textuels pour identifier les profils d'exigences fonctionnelles (FRPs) afin de construire un modèle de variabilité écrit en OVM (Object Variability Model) [NE08]. Les FRPs sont des caractéristiques du système visibles par l'utilisateur écrites sous forme de paires composées d'un verbe suivi de son complément d'objet direct. Les auteurs utilisent diverses techniques pour identifier les unités d'indexation : recherche de verbes, racinisation (ou stemming), étiquetage des parties du discours, suppression des mots vides (ou stop words) et identification des affinités linguistiques sous forme d'unités de 2 mots. Dans un deuxième article, les auteurs examinent l'interaction entre les exigences de qualité et les exigences fonctionnelles à l'aide de l'analyse formelle de concepts [NE09]. À cette fin, ils construisent un tableau où les objets sont des scénarios et les attributs sont les exigences. Le treillis de concepts met alors en évidence et organise les interactions entre les exigences. Une autre approche (FENL) est développée dans [BKSJ16]. Les auteurs utilisent LSA (Latent Semantic Analysis) suivie d'algorithmes de regroupement (dont k-means) pour identifier les caractéristiques à partir des critiques logicielles.Dans notre travail, nous analysons des récits utilisateurs et utilisons un modèle pré-entraîné Word2Vec pour la vectorisation, et le méthode k-means pour le regroupement, après avoir déterminé le nombre de clusters avec la méthode du coude (Elbow). Mefteh et al. [MBB16] utilisent plus spécifiquement des exigences fonctionnelles multilingues, qui sont des phrases courtes telles que « Le système doit bloquer et supprimer les logiciels espions ». Ces exigences sont analysées pour extraire des caractéristiques, en utilisant diverses techniques. Dans notre processus de traitement du langage naturel, nous utilisons le format des récits utilisateurs pour la décomposition et nous séparons le rôle de la caractéristique. Mefteh et al. utilisent

uniquement des exigences fonctionnelles du point de vue du système. Pour la représentation/vectorisation du texte, ils utilisent TFIDF et ajoutent des synonymes provenant d'un thésaurus, tandis que nous utilisons un modèle pré-entraîné word2Vec.

Approches générales

Des algorithmes efficaces ont été développés pour extraire des graphes de caractéristiques à partir de formules en forme CNF (Conjunctive Normal Form) ou DNF (Disjunctive Normal Form) [SRA+14]. Une approche basée sur un algorithme génétique a été conçue par Linsbauer et al. [LLHE14]. L'extraction de modèles de caractéristiques à partir de grandes collections de descriptions partielles de produits a été abordée dans la littérature [DDH+13]. Une méthode combinant une interaction avec un expert pour paramétrer l'extraction et un algorithme automatique qui fusionne les descriptions de produits pour calculer la hiérarchie des caractéristiques et ajouter ensuite la variabilité a été proposée [ACP+12]. La synthèse de modèles de caractéristiques avec attributs a fait l'objet d'une étude publiée dans [BBGA15], tandis que l'utilisation d'ontologies dans le processus a également été explorée [BABN16]. En outre, des modèles de caractéristiques ont été construits à partir d'exigences exprimées sous forme de diagrammes de cas d'utilisation, de scénarios et d'exigences fonctionnelles [MBB18]. Le modèle Promote-pl [KMB20] présente un processus systématique pour l'ingénierie des lignes de produits, ainsi qu'une analyse de ses implications et de ses liens avec les pratiques logicielles. L'objectif est de proposer un modèle réaliste et actuel afin de mieux comprendre comment les organisations développent des plateformes logicielles, en mettant en évidence les divergences entre les modèles historiques et les pratiques industrielles. Rubin et al. [RCC15] détaillent trois cas différents de gestion du clonage, allant des techniques de dérivation utilisées dans différentes entreprises au clonage ad-hoc en tant que compromis avant de migrer complètement vers une LPL (Ligne de Produit Logiciel) pour sa simplicité. ITK se trouve actuellement dans une situation similaire, et notre étude de l'identification de la variabilité constitue une étape vers cette migration. Martinez et al. [MAZ17] ont réalisé un catalogue de cas d'adoption de LPL. L'objectif de ce catalogue est de fournir des exemples similaires aux futurs utilisateurs ou de faciliter l'accès à de futures recherches.

Approches exploitant l'Analyse Formelle de Concepts

L'analyse formelle de concepts (AFC) a été explorée dans plusieurs approches concernant l'extraction des relations logiques ou la synthèse de modèles de caractéristiques. Les utilisations classiques des caractéristiques sont identifiées dans [LP07a]. Ryssel et al. proposent une approche pour extraire des diagrammes de caractéristiques et des contraintes inter-arbres à partir de contextes formels en utilisant l'analyse formelle de concepts [RPK11]. L'approche construit une hiérarchie de concepts et une base d'implications. [AHS+14] développe une approche alternative de synthèse de modèles de

caractéristiques qui structure l'arbre avec des nœuds internes représentant des opérateurs logiques (par exemple un nœud OU) plutôt que des caractéristiques abstraites. Dans [MBB16], après une étape où les caractéristiques sont extraites des exigences fonctionnelles multilingues, l'analyse formelle de concepts et plusieurs heuristiques sont appliquées pour synthétiser un modèle de caractéristiques. Une synthèse des correspondances entre les structures conceptuelles et les modèles de caractéristiques est présentée dans [CHN19d]. Cela permet de caractériser les *classes d'équivalence* des modèles de caractéristiques, selon leur intégration dans une structure conceptuelle, qui est canonique. L'analyse relationnelle de concepts (ARC) a également été exploitée dans la littérature pour extraire des relations logiques dans le contexte des lignes de produits logiciels. Un modèle de variabilité de caractéristiques est extrait dans [NLHS21] en tant que première étape d'une approche qui traite du problème de l'association des caractéristiques et des interactions entre caractéristiques à leurs artefacts de code correspondants. L'ARC est utilisée pour construire les associations et dériver des formules logiques sur les caractéristiques qui annotent le code source. Dans [CHN19a], les auteurs utilisent l'ARC pour analyser la variabilité dans un ensemble de produits interconnectés, par exemple des logiciels de comptabilité (première famille de produits) décrits par le système de gestion de base de données utilisé (deuxième famille de produits), les langages de programmation dans lesquels ils sont écrits (troisième famille de produits), et ainsi de suite. Des pistes pour synthétiser des modèles de caractéristiques avec des références [CHE05] sont proposées.

Notre positionnement

Nous capitalisons sur plusieurs de ces recherches dans notre travail. L'inspiration provient de l'utilisation d'ontologies telles que celles mentionnées dans les références [BABN16] et [CKK06]. Nous incorporons les correspondances établies au fil des années par divers chercheurs entre les structures conceptuelles, les modèles de caractéristiques, et en général, les relations entre les structures conceptuelles et la structuration logique de la variabilité d'une LPL. En comparaison avec les travaux existants qui exploitent l'AFC, notre approche prend en considération la connexion de nombreux artefacts distincts, englobant les récits utilisateurs, les fichiers impactés, les problèmes, les opérations de fusion et les modifications. Notamment, l'utilisation des récits utilisateurs exige la prise en compte des rôles et des caractéristiques associées à ces rôles, ce qui constitue également une caractéristique particulière de notre méthode. Plusieurs techniques ont été élaborées pour extraire automatiquement des modélisations de processus à partir de descriptions textuelles [vdACLR19]. Néanmoins, ces méthodes se concentrent principalement sur la modélisation de processus en négligeant les modèles déclaratifs, qui permettent d'encapsuler efficacement des comportements de processus complexes.

5.7 Conclusion

Dans ce chapitre, nous avons présenté un processus automatisé de synthèse de modèles de caractéristiques à partir de produits logiciels existants construits dans le cadre de projets agiles. En exploitant le traitement du langage naturel, la vectorisation, le regroupement, l'intégration d'ontologies et l'analyse formelle de concepts et l'analyse relationnelle de concepts, nous avons pu identifier et affiner les caractéristiques concrètes et abstraites, ainsi qu'établir des relations de traçabilité entre les récits utilisateurs et le code source. Notre processus repose sur le traitement du langage naturel pour identifier les rôles et les caractéristiques. Avec le traitement du langage naturel, nous regroupons également les caractéristiques en caractéristiques abstraites. Ensuite, nous exploitons l'analyse relationnelle de concepts avec les relations entre nos concepts. Cette approche offre un moyen complet d'extraire des modèles de caractéristiques pertinents à partir de produits existants et peut être facilement modifiée par des experts pour produire le modèle de caractéristiques réel de la ligne de produit.

En perspective de ce travail, nous prévoyons de peaufiner notre processus en effectuant une analyse approfondie du code source et en intégrant les Arbres de Syntaxe Abstraite (AST) pour identifier la variabilité à une granularité plus fine.

En outre, nous prévoyons d'intégrer l'ontologie de différentes manières dans notre processus pour l'améliorer en guidant le regroupement avec les relations ontologiques. Les améliorations attendues concernent l'identification des caractéristiques abstraites et le raffinement de leur organisation hiérarchique. De plus, nous prévoyons d'approfondir l'analyse des règles, en particulier celles ayant un support non nul qui correspondent souvent à des exclusions accidentelles. Nous aimerions également combiner l'utilisation de l'ontologie et du treillis de concepts généré par l'ARC pour identifier des groupes pertinents (alternative, *ou*).

CHAPITRE 6

Conclusion

Sommaire

Dans ce dernier chapitre, nous présentons une synthèse des travaux réalisés dans cette thèse sur l'ingénierie agile des LPL appliquée aux logiciels d'aide à la décision pour l'agriculture. Ensuite nous discutons les perspectives, les travaux futurs et ouvertures possible. Nous concluons par la description des différentes collaborations que ce travail de recherche a permis de mettre en place.

6.1 Synthèse des travaux

Les trois principaux travaux de cette thèse s'inscrivent dans l'objectif d'assister notre partenaire industriel, l'entreprise ITK, dans la migration d'une famille de logiciels vers une LPL. Pour ce faire, nous avons étudié trois axes : l'adoption de la LPL, l'étude de la variabilité dans les schémas de données, et l'étude de la variabilité dans les spécifications agiles. Ces trois axes se retrouvent dans les trois grandes parties de cette thèse.

Évaluation empirique de la perception

Nous avons étudié la perception de la migration de logiciels similaires vers une ligne de produits logiciels. Il s'agit d'un processus coûteux et complexe qui nécessite une préparation et une ouverture d'esprit vis-à-vis du changement et de l'évolution. Des entretiens ont été menés avec les employés de notre partenaire pour évaluer le processus de développement actuel, les avantages et les risques de la migration, du point de vue des acteurs clés du projet. Les acteurs ont pu envisager des avantages dans la migration. La clé pour atténuer les risques réside dans la sensibilisation des parties prenantes, le maintien des bonnes pratiques et leur implication dans le processus de migration pour une transition en douceur. En se basant sur les réponses concernant les pratiques actuelles, nous avons choisi de lancer les travaux de migration en étudiant la variabilité au sein des descriptions des simulateurs.

Identification de la variabilité à partir des schémas de données des simulateurs

Dans un second temps, nous avons examiné les pratiques de développement de notre partenaire. Ils développent des logiciels visant à soutenir la prise de décision dans le domaine de l'agriculture. Ces systèmes incluent des simulateurs pour aider les agriculteurs à prédire le cycle de vie des plantes, avec des paramètres (données) spécifiques pour chaque type de prédiction. Afin de faciliter la création d'une ligne de produits logiciels pour la dérivation de nouveaux produits et la maintenance de la base de code existante, il est nécessaire d'identifier les caractéristiques communes des simulateurs. C'est un processus que nous avons automatisé avec l'aide de l'AFC pour mettre en évidence la variabilité logicielle. Lors du développement d'un nouveau produit destiné à une nouvelle culture, l'étape initiale consiste à intégrer les simulateurs. Après avoir étudié la variabilité de ces simulateurs, nous avons pu amorcer la transition vers l'utilisation d'ITK en identifiant les fonctionnalités pertinentes à l'aide des pratiques de développement à ITK.

Combler le fossé entre les récits utilisateurs et les modèles de caractéristiques en tirant parti des systèmes de contrôle de versions

Ensuite, nous avons élaboré une méthode de synthèse de modèles de caractéristiques qui combine les exigences en agilité pour les logiciels, en mettant l'accent sur les récits utilisateurs, ainsi que les demandes de fusion de code source sur des plateformes de contrôle de versions, ouvertes pour l'implémentation des récits utilisateurs. Nous avons également exploité les ontologies de domaine pour enrichir et mieux organiser ces connaissances. Le point central de cette approche est de considérer les demandes de fusion dans les systèmes de contrôle de versions comme le lien entre les récits utilisateurs (exigences) et le code source (implémentation). Pour parvenir à cette synthèse de modèles de caractéristiques, nous avons combiné plusieurs approches. Tout d'abord,

le traitement du langage naturel et le regroupement des récits utilisateurs qui sont utilisés pour identifier les caractéristiques. Ensuite, l'AFC est employée pour obtenir les règles logiques à partir des résultats des étapes du traitement automatique du langage naturel. Nous avons mis en œuvre et évalué cette méthode sur un ensemble de données provenant de notre partenaire industriel, montrant ainsi son intérêt dans la synthèse de modèles de caractéristiques en vue de la migration de la base de code vers une LPL.

En résumé, notre objectif a été de faciliter la migration vers une LPL en abordant les aspects de perception, de variabilité logicielle et de synthèse des modèles de caractéristiques. Nous avons d'abord évalué la perception de la migration, mettant en évidence l'importance de la sensibilisation des parties prenantes et de leur implication pour une transition réussie. Ensuite, nous avons automatisé l'identification de la variabilité dans les simulateurs. Enfin, notre méthode de synthèse de modèles de caractéristiques a combiné les récits utilisateurs, les demandes de fusion de code source et les ontologies de domaine pour la migration vers une LPL. Ces travaux offrent des solutions pratiques pour notre partenaire et d'autres entreprises qui pourraient être confrontées à des défis similaires dans le domaine du développement logiciel.

6.2 Perspectives

Les différentes contributions pourraient être étendues avec des travaux pouvant se poursuivre à plus ou moins long terme.

Une seconde session d'interview pour comprendre l'après-migration et l'efficacité des méthodes testées. Après avoir étudié la perception des acteurs clés du projet et avoir proposé des lignes directrices, il serait intéressant, une fois le projet terminé, d'étudier la perception de ces lignes directrices, de leur suivi et de leur contribution à la réussite ou à l'échec du projet de migration.

Intégrer la variabilité des simulateurs au sein des modèles de caractéristiques. L'intégration d'un nouveau simulateur marque le début d'un nouveau produit dans le processus de développement d'ITK. C'est pourquoi l'identification de la variabilité dans les schémas d'entrée et de sortie n'est que la première étape. L'intégration de cette variabilité couplée aux modèles de caractéristiques pourrait permettre de générer des configurations à partir du simulateur à intégrer.

Intégrer totalement l'ontologie du domaine pour automatiser complètement la méthode. L'ontologie du domaine est actuellement utilisée pour valider les catégories et aider lors des étapes du traitement automatique du langage naturel. Si cette ontologie était intégrée au processus, l'approche de génération de modèles de caractéristiques

deviendrait totalement automatisée. L'ontologie pourrait servir à améliorer le regroupement des caractéristiques, à nommer les caractéristiques abstraites et à proposer une hiérarchie dans les caractéristiques.

Compléter le processus en séparant le code source pour chaque caractéristique. Le code source associé à chaque caractéristique, une fois isolé, permettrait de générer des variants de la famille de logiciels en fonction du choix des récits utilisateurs. Pour ce faire, il serait possible de s'intéresser aux Arbres de Syntaxe Abstraite (AST) pour identifier la variabilité à une granularité plus fine et ainsi localiser précisément les caractéristiques. Pour valider cela, une preuve de concept consisterait en la génération de variants correspondants aux produits existants.

Utiliser les "bascules de caractéristiques" comme première étape de migration du code. Pour amorcer la migration du code à partir des caractéristiques localisées, il serait possible de commencer une transition en douceur par l'intégration des "bascules de caractéristiques" (*feature toggles*) dans le code source. Cela offrirait la possibilité d'activer ou non certaines caractéristiques dynamiquement, et ainsi améliorer le processus actuel de déploiement de nouveaux produits, avant d'implémenter une approche de LPL plus systématique en combinant caractéristiques statiques et dynamiques, comme décrit dans [JKA22b].

Utiliser des grands modèles de langage (Large Language Models) pour comparer avec les résultats produits par nos méthodes. Depuis peu, les intelligences artificielles génératrices telles que ChatGPT ou BERT sont étudiées pour leur capacité à aider et à guider l'ingénierie logicielle et l'ingénierie des LPL [AM23, ADJ23]. Comparer notre identification de la variabilité au sein des schémas de données à partir des contextes formels pourrait être une première étape. Ensuite, ces méthodes génératives peuvent être utilisées pour la classification des caractéristiques et pour nommer les caractéristiques identifiées.

Ces perspectives visent à renforcer notre capacité à gérer efficacement la migration de logiciels, à améliorer la qualité des produits résultants et à maximiser l'efficacité de notre approche. Les développements futurs dans ces domaines contribueront à une meilleure compréhension des défis liés à la migration de logiciels.

6.3 Contributions et collaborations

Contributions

Certains travaux détaillés dans cette thèse ont été publiés dans des ateliers ou des conférences.

— L'article intitulé "**Variability Extraction from Simulator I/O Data Schemata in Agriculture Decision-Support Software**" [1] [GHK+21], présenté lors de FCA4AI 2021, décrit une méthode d'extraction de la variabilité à partir des schémas de données d'entrée et de sortie des simulateurs dans le contexte des logiciels de notre partenaire industriel, l'entreprise ITK.

— L'article "**Feature and variability extraction from Agile specifications and their related source code for software product line migration**" [Geo22], publié dans le cadre d'un *doctoral symposium* à la conférence SPLC 2022, décrit une approche d'extraction des caractéristiques et de la variabilité à partir des spécifications Agile et de leur code source dans le cadre d'une migration de LPL.

— L'article "**Guiding Feature Models Synthesis from User-Stories : An Exploratory Approach**" [2] [GRH+23], présenté à la conférence VAMOS 2023, décrit les premières étapes de la méthode visant à guider la synthèse de modèles de caractéristiques à partir des récits utilisateurs.

— Le notebook "**From user stories to feature models**" [3] [GH23], présenté lors de l'atelier "*Computational Notebooks for FCA*" (CoNo-Concepts 2023) de la conférence ICFCA'2023 contient le code pour la génération de modèle de caractéristique à partir des récits utilisateur a été présenté à contenant la méthode

Plusieurs autres publications sont en cours de relecture ou sont prévues :

— La première est en cours de relecture et porte sur l'évaluation empirique de la perception de l'ingénierie SPL par une PME avant de migrer sa base de code. Cet article décrit le processus d'interview et d'analyse qui nous a permis de proposer des lignes directrices pour une migration réussie.

— Une publication prévue détaillera le processus pour combler le fossé entre les récits utilisateur et les modèles de caractéristiques en tirant parti des systèmes de contrôle de version dans le cadre de la migration des lignes de produits logiciels.

Collaborations

Au cours des travaux présentés dans cette thèse, nous avons eu l'opportunité d'encadrer plusieurs étudiants dans le but d'explorer et d'enrichir les projets en cours. De plus, nous avons fait des collaborations avec d'autres chercheurs au sein du laboratoire, ce qui nous a permis de renforcer nos approches et d'affiner nos résultats.

Afin d'explorer diverses méthodes de classification de texte et de les appliquer aux récits utilisateurs, nous avons supervisé Liam Rice (en 4e année à Polytech'Montpellier)

1. https://gite.lirmm.fr/tgeorges/interviewartefact
2. https://gite.lirmm.fr/tgeorges/artefacts-guiding-feature-models-synthesis-from-user-stories-an-exploratory-approach
3. https://gite.lirmm.fr/tgeorges/CoNoConcepts_ThomasGEORGES_LIRMM

en juin, juillet et août 2022. Il a effectué des tests et des comparaisons de méthodes telles que LSI ou LDA, et a proposé un pipeline modulaire qui permet de classifier les récits des utilisateurs [4]. Nous avons analysé l'évolution de la variabilité au fil du temps lorsqu'un système évolue, en utilisant la variabilité exprimée par l'AOC-Poset. Dans le cadre de cette étude, nous avons supervisé Richard Picole-Ollivier (en Licence 2 en informatique, Université de Montpellier) en juin et juillet 2022. Sa mission était de mettre en évidence les modifications entre deux AOC-Poset [5]. Nous avons examiné la possibilité d'isoler des caractéristiques à partir de *commits* en explorant l'utilisation des arbres de syntaxe abstraite. Dans le cadre de cette étude, nous avons supervisé Elio Torquet (en Licence 2 en informatique, Université de Montpellier) en juin et juillet 2023. Il a été chargé de sélectionner et comparer deux *commits* afin de repérer les parties de code spécifiques à chaque *commit* et les parties de code partagées [6]. Pour évaluer la généralisation et la viabilité de notre approche, nous avons supervisé Clément Callendret et Paul Deligne (en Licence 2 en informatique, Université de Montpellier) en juin et juillet 2023. Ils se sont penchés sur l'identification des récits utilisateurs et la création d'une gamme de produits. Ce projet avait pour objectif de décomposer le logiciel libre "*Broadleaf Commerce*". Afin de compléter notre méthode, nous avons utilisé l'outil Mobioos pour repérer les caractéristiques et élaborer une gamme de produits logiciels [7]. Pendant notre analyse des règles de l'AFC et de l'ARC pour l'identification des contraintes et construire les modèles de caractéristiques, nous avons collaboré avec Julie Cailler et Alexandre Bazin. Alexandre Bazin a contribué à la sélection et à l'utilisation des bases de règles, tandis que Julie Cailler a apporté son expertise dans les règles logiques et leur interprétation.

Grâce à ces collaborations, nous avons pu explorer et enrichir nos travaux avec diverses méthodes de classification de texte, d'analyse de la variabilité logicielle, d'extraction de caractéristiques, et d'analyse des règles.

4. https://gite.lirmm.fr/tgeorges/User_Story_Text_Mining
5. https://gite.lirmm.fr/tgeorges/Comparaison_AOC-Poset
6. https://gite.lirmm.fr/tgeorges/Compaison-of-feature-implementation-AST
7. https://gite.lirmm.fr/tgeorges/MobioosApplicationToBroadleafCommerceDemo

Bibliographie

[AAF+09] Nour Alhouda Aboud, Gabriela Arévalo, Jean-Rémy Falleri, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. Automated architectural component classification using concept lattices. In Joint Working IEEE/IFIP Conference on Software Architecture 2009 and European Conference on Software Architecture 2009, WICSA/ECSA 2009, Cambridge, UK, 14-17 September 2009, pages 21–30. IEEE Computer Society, 2009.

[ABS+22] Ra'Fat Al-Msie'deen, Anas H. Blasi, Hamzeh Eyal Salman, Saqer Saleh Alja'afreh, Ahmad Abadleh, Mohammed A. Alsuwaiket, Awni Hammouri, Asmaa Jameel Al-Nawaiseh, Wafa Tarawneh, and Suleyman A. Al-Showarah. Detecting commonality and variability in use-case diagram variants. CoRR, abs/2203.00312, 2022.

[AC10] Faheem Ahmed and Luiz Fernando Capretz. An organizational maturity model of software product line engineering. Softw. Qual. J., 18(2) :195–225, 2010.

[AC15a] Faheem Ahmed and Luiz Fernando Capretz. A business maturity model of software product line engineering. CoRR, abs/1507.06897, 2015.

[AC15b] Faheem Ahmed and Luiz Fernando Capretz. A framework for process assessment of software product line. CoRR, abs/1507.06948, 2015.

[ACP+12] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On extracting feature models from product descriptions. In Ulrich W. Eisenecker, Sven Apel, and Stefania Gnesi, editors, Sixth International

Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings, pages 45–54. ACM, 2012.

[ADH+11] Zeina Azmeh, Maha Driss, Fady Hamoui, Marianne Huchard, Naouel Moha, and Chouki Tibermacine. Selection of composable web services driven by user requirements. In IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011, pages 395–402. IEEE Computer Society, 2011.

[ADJ23] Mathieu Acher, José Ángel Galindo Duarte, and Jean-Marc Jézéquel. On programming variability with large language model-based assistant. In Paolo Arcaini, Maurice H. ter Beek, Gilles Perrouin, Iris Reinhartz-Berger, Miguel R. Luaces, Christa Schwanninger, Shaukat Ali, Mahsa Varshosaz, Angelo Gargantini, Stefania Gnesi, Malte Lochau, Laura Semini, and Hironori Washizaki, editors, Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023, pages 8–14. ACM, 2023.

[ADN03] Gabriela Arévalo, Stéphane Ducasse, and Oscar Nierstrasz. Xray views : Understanding the internals of classes. In 18th IEEE International Conference on Automated Software Engineering (ASE 2003), 6-10 October 2003, Montreal, Canada, pages 267–270. IEEE Computer Society, 2003.

[AGM+12] Bastien Amar, Abdoulkader Osman Guédi, André Miralles, Marianne Huchard, Thérèse Libourel, and Clémentine Nebut. Using formal concept analysis to extract a greatest common model. In Leszek A. Maciaszek, Alfredo Cuzzocrea, and José Cordeiro, editors, ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 1, Wroclaw, Poland, 28 June - 1 July, 2012, pages 27–37. SciTePress, 2012.

[AHS+14] Ra'Fat Al-Msie'deen, Marianne Huchard, Abdelhak Seriai, Christelle Urtado, and Sylvain Vauttier. Reverse engineering feature models from software configurations using formal concept analysis. In Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014., pages 95–106, 2014.

[AJL+20] Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, and Daniel Sundmark. Product line adoption in industry : an experience report from the railway domain. In Roberto Erick Lopez-Herrejon, editor, SPLC '20 : 24th

ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A, pages 3 :1–3 :11. ACM, 2020.

[ALB+07] Sven Apel, Christian Lengauer, Don Batory, Bernhard Möller, and Christian Kästner. An algebra for feature-oriented software development. 01 2007.

[ALL+17] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. Reengineering legacy applications into software product lines : a systematic mapping. Empir. Softw. Eng., 22(6) :2972–3016, 2017.

[AM23] Mathieu Acher and Jabier Martinez. Generative AI for reengineering variants into software product lines : An experience report. In Paolo Arcaini, Maurice H. ter Beek, Gilles Perrouin, Iris Reinhartz-Berger, Ivan Machado, Silvia Regina Vergilio, Rick Rabiser, Tao Yue, Xavier Devroey, Mónica Pinto, and Hironori Washizaki, editors, Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023, pages 57–66. ACM, 2023.

[ÅNKB19] Jonas Åkesson, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger. Migrating the android apo-games into an annotation-based software product line. In Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi, editors, Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019, pages 19 :1–19 :5. ACM, 2019.

[APAF21] Inmaculada Ayala, Alessandro V. Papadopoulos, Mercedes Amor, and Lidia Fuentes. Prodspl : Proactive self-adaptation based on dynamic software product lines. J. Syst. Softw., 175 :110909, 2021.

[AV14] Wesley Klewerton Guez Assunção and Silvia Regina Vergilio. Feature location for software product line migration : a mapping study. In Stefania Gnesi, Alessandro Fantechi, Maurice H. ter Beek, Goetz Botterweck, and Martin Becker, editors, 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers, SPLC '14, Florence, Italy, September 15-19, 2014, pages 52–59. ACM, 2014.

[BA05] Kent L. Beck and Cynthia Andres. Extreme programming explained - embrace change, Second Edition. The XP series. Addison-Wesley, 2005.

[BABN16] Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr. Breathing ontological knowledge into feature model synthesis : an empirical study. Empir. Softw. Eng., 21(4) :1794–1841, 2016.

[Bat06] D Batory. Feature modularity for product-lines. tutorial at : Oopsla'06 generative programming and component engineering (gpce). 10 2006.

[Baz18] Alexandre Bazin. A depth-first search algorithm for computing pseudo-closed sets. Discret. Appl. Math., 249 :28–35, 2018.

[BB11] Jan Bosch and Petra Bosch-Sijtsema. Introducing agile customer-centered development in a legacy software product line. Softw. Pract. Exp., 41(8) :871–882, 2011.

[BBC06] Don S. Batory, David Benavides, and Antonio Ruiz Cortés. Automated analysis of feature models : challenges ahead. Commun. ACM, 49(12) :45–47, 2006.

[BBGA15] Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher. Synthesis of attributed feature models from product descriptions. In Douglas C. Schmidt, editor, Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pages 1–10. ACM, 2015.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001.

[BC05] Felix Bachmann and Paul Clements. Variability in software product lines. Technical Report CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[BCK03] Ross Buhrdorf, Dale Churchett, and Charles W. Krueger. Salion's experience with a reactive software product line approach. In Frank van der Linden, editor, Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers, volume 3014 of Lecture Notes in Computer Science, pages 317–322. Springer, 2003.

[BdMSNdAdLM11] Jonatas Ferreira Bastos, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira.

Adopting software product lines : A systematic mapping study. In 15th International Conference on Evaluation & Assessment in Software Engineering, EASE 2011, Durham, UK, 11-12 April 2011, Proceedings, pages 11–20. IET - The Institute of Engineering and Technology / IEEE Xplore, 2011.

[BdMSNO+17] Jonatas Ferreira Bastos, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Software product lines adoption in small organizations. J. Syst. Softw., 131 :112–128, 2017.

[Beu12] Danilo Beuche. Modeling and building software product lines with pure : : variants. In Eduardo Santana de Almeida, Christa Schwanninger, and David Benavides, editors, 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012, Volume 2, page 255. ACM, 2012.

[BFG+01] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In Frank van der Linden, editor, Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers, volume 2290 of Lecture Notes in Computer Science, pages 13–21. Springer, 2001.

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse : A methodology to develop software product lines. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability, SSR 1999, May 21-23, 1999, Los Angeles, CA, USA, pages 122–131. ACM, 1999.

[BGH+14] Anne Berry, Alain Gutierrez, Marianne Huchard, Amedeo Napoli, and Alain Sigayret. Hermes : a simple and efficient algorithm for building the aoc-poset of a binary relation. Ann. Math. Artif. Intell., 72(1-2) :45–71, 2014.

[BHM+23] Giovanna Bettin, Julio Herculani, Amanda Melo, Luiz C. M. Andrade, and Edson OliveiraJr. Efficacy, efficiency and effectiveness of smarty-based software product line inspection techniques : A controlled quasi-experiment. In Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS '23, page 40–49, New York, NY, USA, 2023. Association for Computing Machinery.

[BIP09] Muhammad Ali Babar, Tuomas Ihme, and Minna Pikkarainen. An industrial case of exploiting product line architectures in agile soft-

ware development. In Dirk Muthig and John D. McGregor, editors, Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24-28, 2009, Proceedings, volume 446 of ACM International Conference Proceeding Series, pages 171–179. ACM, 2009.

[BKS15] Noor Hasrina Bakar, Zarinah M. Kasirun, and Norsaremah Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines : A systematic literature review. Journal of Systems and Software, 106 :132–149, 2015.

[BKSJ16] Noor Hasrina Bakar, Zarinah M. Kasirun, Norsaremah Salleh, and Hamid A. Jalab. Extracting features from online software reviews to aid requirements reuse. Applied Soft Computing, 49 :1297–1315, 2016.

[BLN86] Carlo Batini, Maurizio Lenzerini, and Shamkant Bhalchandra Navathe. A comparative analysis of methodologies for database schema integration. ACM Computer Survey, 18 :323–364, 1986.

[BMK+02] Günter Böckle, Jesús Bermejo Muñoz, Peter Knauber, Charles W. Krueger, Julio Cesar Sampaio do Prado Leite, Frank van der Linden, Linda M. Northrop, Michael E. Stark, and David M. Weiss. Adopting and institutionalizing a product line culture. In Gary J. Chastek, editor, Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19-22, 2002, Proceedings, volume 2379 of Lecture Notes in Computer Science, pages 49–59. Springer, 2002.

[BNR+14] Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wasowski. Three cases of feature-based variability modeling in industry. In Jürgen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfrán, editors, Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings, volume 8767 of Lecture Notes in Computer Science, pages 302–319. Springer, 2014.

[BNSM15] Tanmay Bhowmik, Nan Niu, Juha Savolainen, and Anas Mahmoud. Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering. Requirements Engineering, 20, 04 2015.

[Boc07] Hans-Hermann Bock. Clustering Methods : A History of k-Means Algorithms, pages 161–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Boe10] Barry W. Boehm. Some future software engineering opportunities and challenges. In Sebastian Nanz, editor, The Future of Software Engineering, pages 1–32. Springer, 2010.

[BONB22] John Businge, Moses Openja, Sarah Nadi, and Thorsten Berger. Reuse and maintenance practices among divergent forks in three software ecosystems. Empir. Softw. Eng., 27(2) :54, 2022.

[Bos00] Jan Bosch. Design and use of software architectures - adopting and evolving a product-line approach. Addison-Wesley, 2000.

[BPL05] Günter Böckle, Klaus Pohl, and Frank Linden. A Framework for Software Product Line Engineering, pages 19–38. 01 2005.

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. A survey of variability modeling in industrial practice. In Stefania Gnesi, Philippe Collet, and Klaus Schmid, editors, The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa , Italy, January 23 - 25, 2013, pages 7 :1–7 :8. ACM, 2013.

[Brö18] Alfred Bröckers. Variability in standard software products - introducing software product line engineering totheinsurance industry. In Volker Gruhn and Rüdiger Striemer, editors, The Essence of Software Engineering, pages 91–105. Springer, 2018.

[BRZ05] Barry Boehm, Hans Dieter Rombach, and Marvin V. Zelkowitz. Foundations of Empirical Software Engineering : The Legacy of Victor R. Basili. Springer-Verlag, Berlin, Heidelberg, 2005.

[BT05] Barry W. Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. IEEE Softw., 22(5) :30–39, 2005.

[BWZ15] Leonard J. Bass, Ingo M. Weber, and Liming Zhu. DevOps - A Software Architect's Perspective. SEI series in software engineering. Addison-Wesley, 2015.

[Car18a] Jessie Carbonnel. L'analyse formelle de concepts : un cadre structurel pour l'étude de la variabilité de familles de logiciels. Theses, Université Montpellier, October 2018.

[Car18b] Jessie Carbonnel. L'analyse formelle de concepts : un cadre structurel pour l'étude de la variabilité de familles de logiciels. (Formal concept analysis : a structural framework to study variability in software families). PhD thesis, University of Montpellier, France, 2018.

[Cat09] Cagatay Catal. Barriers to the adoption of software product line engineering. ACM SIGSOFT Softw. Eng. Notes, 34(6) :1–4, 2009.

[CBA09] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines : a systematic review. In Dirk Muthig and John D. McGregor, editors, Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24-28, 2009, Proceedings, volume 446 of ACM International Conference Proceeding Series, pages 81–90. ACM, 2009.

[CBHN20] Jessie Carbonnel, Karell Bertet, Marianne Huchard, and Clémentine Nebut. FCA for software product line representation : Mixing configuration and feature relationships in a unique canonical representation. Discret. Appl. Math., 273 :43–64, 2020.

[CDFR08] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Formal concept analysis enhances fault localization in software. In Raoul Medina and Sergei A. Obiedkov, editors, Formal Concept Analysis, 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008, Proceedings, volume 4933 of Lecture Notes in Computer Science, pages 273–288. Springer, 2008.

[CFM19] Daniel Cruz, Eduardo Figueiredo, and Jabier Martinez. A literature review and comparison of three feature location techniques using argouml-spl. In Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[Cha96] Patrick Y. K. Chau. An empirical assessment of a modified technology acceptance model. J. Manag. Inf. Syst., 13(2) :185–204, 1996.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. Softw. Process. Improv. Pract., 10(2) :143–169, 2005.

[CHN18] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Towards the Extraction of Variability Information to Assist Variability Modelling of Complex Product Lines. In Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems - VAMOS 2018, pages 113–120, Madrid, Spain, 2018. ACM Press.

[CHN19a] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Exploring the variability of interconnected product families with relational concept analysis. In Carlos Cetina, Oscar Díaz, Laurence Du-

chien, Marianne Huchard, Rick Rabiser, Camille Salinesi, Christoph Seidl, Xhevahire Tërnava, Leopoldo Teixeira, Thomas Thüm, and Tewfik Ziadi, editors, Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019, pages 90 :1–90 :8. ACM, 2019.

[CHN19b] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. J. Syst. Softw., 152 :1–23, 2019.

[CHN19c] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Towards complex product line variability modelling : Mining relationships from non-boolean descriptions. J. Syst. Softw., 156 :341–360, 2019.

[CHN19d] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. Journal of Systems and Software, 152 :1–23, 2019.

[CHS08] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a feature : A requirements engineering perspective. In José Luiz Fiadeiro and Paola Inverardi, editors, Fundamental Approaches to Software Engineering, 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, volume 4961 of Lecture Notes in Computer Science, pages 16–30. Springer, 2008.

[CKK06] Krzysztof Czarnecki, Chang Hwan Peter Kim, and Karl Trygve Kalleberg. Feature models are views on ontologies. In Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings, pages 41–51. IEEE Computer Society, 2006.

[CLC04] David Cohen, Mikael Lindvall, and Patricia Costa. An introduction to agile methods. Adv. Comput., 62 :1–66, 2004.

[CLPP23] Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, and Ángeles Saavedra Places. Adapting the database to feature changes in software product lines. In Paolo Arcaini, Maurice H. ter Beek, Gilles Perrouin, Iris Reinhartz-Berger, Miguel R. Luaces, Christa Schwanninger, Shaukat Ali, Mahsa Varshosaz, Angelo Gargantini, Stefania Gnesi, Malte Lochau, Laura Semini, and Hironori Washizaki,

editors, Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023, pages 194–200. ACM, 2023.

[CN02] Paul Clements and Linda M. Northrop. Software product lines - practices and patterns. SEI series in software engineering. Addison-Wesley, 2002.

[CnKL+23] Alejandro Cortiñas, Jacob Krüger, Victor Lamas, Miguel R. Luaces, and Oscar Pedreira. How to retire and replace a software product line. In Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A, SPLC '23, page 275–286, New York, NY, USA, 2023. Association for Computing Machinery.

[Coc04] Alistair Cockburn. Crystal clear a human-powered methodology for small teams. 01 2004.

[Coh04] Mike Cohn. User Stories Applied : For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., USA, 2004.

[Cou96] Patrick Cousot. Abstract interpretation. ACM Comput. Surv., 28(2) :324–328, 1996.

[CS09] Paul C. Clements and Mary Shaw. "the golden age of software architecture" revisited. IEEE Softw., 26(4) :70–72, 2009.

[Cza02] Krzysztof Czarnecki. Generative programming : Methods, techniques, and applications. In Cristina Gacek, editor, Software Reuse : Methods, Techniques, and Tools, 7th International Conference, ICSR-7, Austin, TX, USA, April 15-19, 2002, Proceedings, volume 2319 of Lecture Notes in Computer Science, pages 351–352. Springer, 2002.

[CZZM05] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. In 13th IEEE International Conference on Requirements Engineering (RE 2005), 29 August - 2 September 2005, Paris, France, pages 31–40. IEEE Computer Society, 2005.

[DDH+13] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In Bertrand Meyer, Luciano Baresi, and Mira Mezini, editors, Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software

Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013, pages 290–300. ACM, 2013.

[DH98] Marie-Catherine Daniel-Vatonne and C. Hemce. On a tree-like representation for symbolic-numeric data and its use in galois lattice method. In Proceedings of 18th International Conference of the Chilean Computer Science Society (SCCC '98), November 12-14, 1998, Antofagasta, Chile, pages 48–57. IEEE Computer Society, 1998.

[DHH+04] Michel Dao, Marianne Huchard, Mohamed Rouane Hacene, Cyril Roume, and Petko Valtchev. Improving generalization level in UML models iterative cross generalization in practice. In Karl Erich Wolff, Heather D. Pfeiffer, and Harry S. Delugach, editors, Conceptual Structures at Work : 12th International Conference on Conceptual Structures, ICCS 2004, Huntsville, AL, USA, July 19-23, 2004. Proceedings, volume 3127 of Lecture Notes in Computer Science, pages 346–360. Springer, 2004.

[DLBH13] Xavier Dolques, Florence Le Ber, and Marianne Huchard. AOC-posets : a scalable alternative to Concept Lattices for Relational Concept Analysis. In CLA : Concept Lattices and their Applications, pages 129–140, La Rochelle, France, October 2013.

[DLCL23] Aleksandar S. Dimovski, Sami Lazreg, Maxime Cordy, and Axel Legay. Family-based model checking of fmultiltl properties. In Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A, SPLC '23, page 41–51, New York, NY, USA, 2023. Association for Computing Machinery.

[DPAG11] Jessica Díaz, Jennifer Pérez, Pedro Pablo Alarcón, and Juan Garbajosa. Agile product line engineering - a systematic literature review. Softw. Pract. Exp., 41(8) :921–941, 2011.

[DRB+13] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of cloning in industrial software product lines. In Anthony Cleve, Filippo Ricca, and Maura Cerioli, editors, 17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013, pages 25–34. IEEE Computer Society, 2013.

[DRGP13] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code : a taxonomy and survey. J. Softw. Evol. Process., 25(1) :53–95, 2013.

[dSdMSNO+11] Ivonei Freitas da Silva, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero

de Lemos Meira. Agile software product lines : a systematic mapping study. Softw. Pract. Exp., 41(8) :899–920, 2011.

[dSdMSNO+14] Ivonei Freitas da Silva, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Software product line scoping and requirements engineering in a small and medium-sized enterprise : An industrial case study. J. Syst. Softw., 88 :189–206, 2014.

[DvDP18] Nicolas Dintzner, Arie van Deursen, and Martin Pinzger. FEVER : an approach to analyze feature-oriented changes and artefact co-evolution in highly configurable systems. Empir. Softw. Eng., 23(2) :905–952, 2018.

[Eck04] Jutta Eckstein. Agile software development in the large - diving into the deep. Dorset House Publishing, 2004.

[EKS03] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. IEEE Trans. Software Eng., 29(3) :210–224, 2003.

[EPPC21] Jorge Echeverría, Francisca Pérez, José Ignacio Panach, and Carlos Cetina. An empirical study of performance using clone & own and software product lines in an industrial context. Inf. Softw. Technol., 130 :106444, 2021.

[ES03] C. Ebert and M. Smouts. Tricks and traps of initiating a product line concept in existing products. In 25th International Conference on Software Engineering, 2003. Proceedings., pages 520–525, 2003.

[FH04] Claudia Fritsch and Ralf Hahn. Product line potential analysis. In Robert L. Nord, editor, Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, volume 3154 of Lecture Notes in Computer Science, pages 228–237. Springer, 2004.

[FLLE14] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Enhancing clone-and-own with systematic reuse for developing software variants. In 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, pages 391–400. IEEE Computer Society, 2014.

[FLLE15] Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. The ECCO tool : Extraction and composition for clone-and-own. In Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum, editors, 37th IEEE/ACM International Conference

on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2, pages 665–668. IEEE Computer Society, 2015.

[FSK+16] Thomas Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. Ten years of product line engineering at danfoss : lessons learned and way ahead. In Hong Mei, editor, Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016, pages 252–261. ACM, 2016.

[GAM10] Yaser Ghanam, Darren Andreychuk, and Frank Maurer. Reactive variability management in agile software development. In Sallyann Freudenberg and Joseph Chao, editors, 2010 Agile Conference, AGILE 2010, Orlando, Florida, USA, August 9-13, 2010, pages 27–34. IEEE Computer Society, 2010.

[GD86] J. L. Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. Mathématiques et Sciences Humaines, 95 :5–18, 1986.

[Geo22] Thomas Georges. Feature and variability extraction from agile specifications and their related source code for software product line migration. In Alexander Felfernig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Clément Quinton, Jianmei Guo, Klaus Schmid, Marianne Huchard, Inmaculada Ayala, José Miguel Rojas, Viet-Man Le, and José Miguel Horcas, editors, SPLC '22 : 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume B, pages 29–33. ACM, 2022.

[GH23] Thomas Georges and Marianne Huchard. From user stories to feature models, *Workshop "Computational Notebooks for FCA" (CoNo-Concepts 2023)*. ICFCA, 2023.

[GHK+21] Thomas Georges, Marianne Huchard, Mélanie König, Clémentine Nebut, and Chouki Tibermacine. Variability extraction from simulator i/o data schemata in agriculture decision-support software. In Proceedings of the 9th Workshop (FCA4AI), co-located with IJCAI 2021, Montréal, Canada, August 2021. Virutal Event, 2021.

[GHM22] Alain Gutierrez, Marianne Huchard, and Pierre Martin. FCA4J : A java library for relational concept analysis and formal concept analysis. In Pablo Cordero and Ondrej Krídlo, editors, Proceedings of the Sixteenth International Conference on Concept Lattices and Their Applications (CLA 2022) Tallinn, Estonia, June 20-22, 2022.,

Tallinn, Estonia, June 20-22, 2022, volume 3308 of CEUR Workshop Proceedings, pages 207–212. CEUR-WS.org, 2022.

[GK01] Bernhard Ganter and Sergei O. Kuznetsov. Pattern Structures and Their Projections. In 9th Int. Conference ICCS'01, Stanford, CA, USA, pages 129–142, 2001.

[GM93] Robert Godin and Hafedh Mili. Building and maintaining analysis-level class hierarchies using galois lattices. In Timlynn Babitsky and Jim Salmons, editors, Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 1993, Washington, DC, USA, September 26 - October 1, 1993, pages 394–410. ACM, 1993.

[GM08] Yaser Ghanam and Frank Maurer. An iterative model for agile product line engineering. In Steffen Thiel and Klaus Pohl, editors, Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops), pages 377–384. Lero Int. Science Centre, University of Limerick, Ireland, 2008.

[GMNR05] Alain Gély, Raoul Medina, Lhouari Nourine, and Yoan Renaud. Uncovering and reducing hidden combinatorics in guigues-duquenne bases. In Bernhard Ganter and Robert Godin, editors, Formal Concept Analysis, Third International Conference, ICFCA 2005, Lens, France, February 14-18, 2005, Proceedings, volume 3403 of Lecture Notes in Computer Science, pages 235–248. Springer, 2005.

[GRH+23] Thomas Georges, Liam Rice, Marianne Huchard, Mélanie König, Clémentine Nebut, and Chouki Tibermacine. Guiding feature models synthesis from user-stories : An exploratory approach. In Myra B. Cohen, Thomas Thüm, and Jacopo Mauro, editors, Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS 2023, Odense, Denmark, January 25-27, 2023, pages 65–70. ACM, 2023.

[GW99] Bernhard Ganter and Rudolf Wille. Formal Concept Analysis - Mathematical Foundations. Springer, 1999.

[GZ22] Karim Ghallab and Tewfik Ziadi. Implementing software product lines with mobioos forge, *Tool Demo Track - Reuse and Software Quality - 20th International Conference on Software and Systems Reuse, Montpellier, France, June.* ICSR, 2022.

[Han11] Geir Kjetil Hanssen. Agile software product line engineering : enabling factors. Softw. Pract. Exp., 41(8) :883–897, 2011.

[HHNV13] Mohamed Rouane Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. Relational concept analysis : mining concept lattices from multi-relational data. Ann. Math. Artif. Intell., 67(1) :81–108, 2013.

[Hir08] Kenji Hiranabe. Kanban applied to software development : From agile to lean. InfoQ. URL : http ://www. infoq. com/articles/hiranabe-lean-agile-kanban [accessed 7 March 2013], 2008.

[HJ04] Jacob Gorm Hansen and Eric Jul. Self-migration of operating systems. In Yolande Berbers and Miguel Castro, editors, Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22, 2004, page 23. ACM, 2004.

[HKM06] William A. Hetrick, Charles W. Krueger, and Joseph G. Moore. Incremental return on incremental investment : Engenio's transition to software product line practice. In Peri L. Tarr and William R. Cook, editors, Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA, pages 798–804. ACM, 2006.

[Hla22] Nicolas Hlad. IsiSPL : un processus automatisé pour faciliter l'ingénierie des lignes de produits logiciels selon une stratégie d'adoption industrielle réactive ou extractive. (IsiSPL : an automated process to facilitate the engineering of software product lines according to a reactive or extractive industrial adoption strategy). PhD thesis, University of Montpellier, France, 2022.

[HLHS21] Nicolas Hlad, Bérénice Lemoine, Marianne Huchard, and Abdelhak-Djamel Seriai. Leveraging relational concept analysis for automated feature location in software product lines. In Eli Tilevich and Coen De Roover, editors, GPCE '21 : Concepts and Experiences, Chicago, IL, USA, October 17 - 18, 2021, pages 170–183. ACM, 2021.

[HLV99] Geert-Jan Houben, K. Lenie, and Koen Vanhoof. A knowledge-based swot-analysis system as an instrument for strategic planning in small and medium sized enterprises. Decis. Support Syst., 26(2) :125–135, 1999.

[HMSS17] Philipp Hohl, Jürgen Münch, Kurt Schneider, and Michael Stupperich. Real-life challenges on agile software product lines in automotive. In Michael Felderer, Daniel Méndez Fernández, Burak Turhan, Marcos Kalinowski, Federica Sarro, and Dietmar

Winkler, editors, Product-Focused Software Process Improvement - 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29 - December 1, 2017, Proceedings, volume 10611 of Lecture Notes in Computer Science, pages 28–36. Springer, 2017.

[HSD21] Nicolas Hlad, Abdelhak-Djamel Seriai, and Christophe Dony. Isispl : Toward an automated reactive approach to build software product lines. CoRR, abs/2107.00552, 2021.

[HSNK22] Andrea Hillenbrand, Uta Störl, Shamil Nabiyev, and Meike Klettke. Self-adapting data migration in the context of schema evolution in nosql databases. Distributed Parallel Databases, 40(1) :5–25, 2022.

[HYCM09] Geir Kjetil Hanssen, Aiko Fallas Yamashita, Reidar Conradi, and Leon Moonen. Maintenance and agile development : Challenges, opportunities and future directions. In 25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada, pages 487–490. IEEE Computer Society, 2009.

[Int20] The Standish Group International. Chaos report, 2020.

[IRW16] Nili Itzik, Iris Reinhartz-Berger, and Yair Wand. Variability analysis of requirements : Considering behavioral differences and reflecting stakeholders' perspectives. IEEE Trans. Software Eng., 42(7) :687–706, 2016.

[JKA22a] Jean-Marc Jézéquel, Jörg Kienzle, and Mathieu Acher. From feature models to feature toggles in practice. In Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A, SPLC '22, page 234–244, New York, NY, USA, 2022. Association for Computing Machinery.

[JKA22b] Jean-Marc Jézéquel, Jörg Kienzle, and Mathieu Acher. From feature models to feature toggles in practice. In Alexander Felfernig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Andreas A. Falkner, Maider Azanza, Miguel Á. Rodríguez Luaces, Megha Bhushan, Laura Semini, Xavier Devroey, Cláudia Maria Lima Werner, Christoph Seidl, Viet-Man Le, and José Miguel Horcas, editors, SPLC '22 : 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A, pages 234–244. ACM, 2022.

[JNL16] Stephen Jones, Joost Noppen, and Fiona Lettice. Management challenges for devops adoption within UK smes. In Danilo Ardagna, Giuliano Casale, André van Hoorn, and Felix Willnecker, editors, Proceedings of the 2nd International Workshop on Quality-Aware

DevOps, QUDOS@ISSTA 2016, Saarbrücken, Germany, July 21, 2016, pages 7–11. ACM, 2016.

[JO09] Meena Jha and Liam O'Brien. Identifying issues and concerns in software reuse in software product lines. In Stephen H. Edwards and Gregory Kulczycki, editors, Formal Foundations of Reuse and Domain Engineering, 11th International Conference on Software Reuse, ICSR 2009, Falls Church, VA, USA, September 27-30, 2009. Proceedings, volume 5791 of Lecture Notes in Computer Science, pages 181–190. Springer, 2009.

[KB20] Jacob Krüger and Thorsten Berger. Activities and costs of re-engineering cloned variants into an integrated platform. In Maxime Cordy, Mathieu Acher, Danilo Beuche, and Gunter Saake, editors, VaMoS '20 : 14th International Working Conference on Variability Modelling of Software-Intensive Systems, Magdeburg Germany, February 5-7, 2020, pages 21 :1–21 :10. ACM, 2020.

[KB21] Jacob Krüger and Thorsten Berger. An empirical analysis of the costs of clone- and platform-oriented software reuse. In Anne Koziolek, Ina Schaefer, and Christoph Seidl, editors, Software Engineering 2021, Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26. Februar 2021, Braunschweig/Virtuell, volume P-310 of LNI, pages 69–70. Gesellschaft für Informatik e.V., 2021.

[KC14] Charles W. Krueger and Paul C. Clements. Systems and software product line engineering with gears from biglever software. In Stefania Gnesi, Alessandro Fantechi, Maurice H. ter Beek, Goetz Botterweck, and Martin Becker, editors, 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers, SPLC '14, Florence, Italy, September 15-19, 2014, pages 121–125. ACM, 2014.

[KCH+90a] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 01 1990.

[KCH+90b] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, 1990.

[KHPS19] Jil Ann-Christin Klünder, Philipp Hohl, Nils Prenner, and Kurt Schneider. Transformation towards agile software product line engineering in large companies : A literature review. J. Softw. Evol. Process., 31(5), 2019.

[KKL+98] Kyo Chul Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM : A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng., 5 :143–168, 1998.

[KL04] Eric S. Knowles and Jay A. Linn. Approach-avoidance model of persuasion : Alpha and omega strategies for change. In Eric S. Knowles and Jay A. Linn, editors, Resistance and Persuasion, pages 117–148. Lawrence Erlbaum Associates, Mahwah, New Jersey, 2004.

[KMB20] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. Promote-pl : a round-trip engineering process model for adopting and evolving product lines. In Roberto Erick Lopez-Herrejon, editor, SPLC '20 : 24th ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A, pages 2 :1–2 :12. ACM, 2020.

[KP08] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Personal opinion surveys. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 63–92. Springer, 2008.

[Kru92] Charles W. Krueger. Software reuse. ACM Comput. Surv., 24(2) :131–183, 1992.

[Kru01] Charles W. Krueger. Easing the transition to software mass customization. In Frank van der Linden, editor, Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers, volume 2290 of Lecture Notes in Computer Science, pages 282–293. Springer, 2001.

[KSG06] Michael Kircher, Christa Schwanninger, and Iris Groher. Transitioning to a software product family approach - challenges and best practices. In Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings, pages 163–171. IEEE Computer Society, 2006.

[KSH08] Pasi Kuvaja, Jouni Similä, and Hanna Hanhela. Software product line adoption - guidelines from a case study. In Zbigniew Huzar, Radek Kocí, Bertrand Meyer, Bartosz Walter, and Jaroslav Zendulka, editors, Software Engineering Techniques - Third IFIP TC 2 Central and East European Conference, CEE-SET 2008, Brno, Czech Republic, October 13-15, 2008, Revised Selected Papers, volume 4980 of Lecture Notes in Computer Science, pages 143–157. Springer, 2008.

[Lar02] Craig Larman. Agile processes and modeling. In Mehmet Aksit, Mira Mezini, and Rainer Unland, editors, Objects, Components, Architectures, Services, and Applications for a Networked World, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7-10, 2002, Revised Papers, volume 2591 of Lecture Notes in Computer Science, page 2. Springer, 2002.

[LBTN15] Artuur Leeuwenberg, Aleksey Buzmakov, Yannick Toussaint, and Amedeo Napoli. Exploring pattern structures of syntactic trees for relation extraction. In Jaume Baixeries, Christian Sacarea, and Manuel Ojeda-Aciego, editors, Formal Concept Analysis - 13th International Conference, ICFCA 2015, Nerja, Spain, June 23-26, 2015, Proceedings, volume 9113 of Lecture Notes in Computer Science, pages 153–168. Springer, 2015.

[Lef07] Dean Leffingwell. Scaling Software Agility : Best Practices for Large Enterprises. The Agile Software Development Series. Addison-Wesley Professional, 2007.

[LFU+22] Viet-Man Le, Alexander Felfernig, Mathias Uta, Thi Ngoc Trang Tran, and Cristian Vidal Silva. Wipeoutr : Automated redundancy detection for feature models. In Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A, SPLC '22, page 164–169, New York, NY, USA, 2022. Association for Computing Machinery.

[LKHB21] Robert Lindohf, Jacob Krüger, Erik Herzog, and Thorsten Berger. Software product-line evaluation in the large. Empirical Software Engineering, 26, 03 2021.

[LLHE14] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Feature model synthesis with genetic programming. In Claire Le Goues and Shin Yoo, editors, Search-Based Software Engineering, pages 153–167, Cham, 2014. Springer International Publishing.

[LM14] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. CoRR, abs/1405.4053, 2014.

[LP07a] Felix Loesch and Erhard Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. In 11th European Conference on Software Maintenance and Reengineering, Software Evolution in Complex Software Intensive Systems, CSMR 2007, 21-23 March 2007, Amsterdam, The Netherlands, pages 159–170, 2007.

[LP07b] Felix Lösch and E. Plödereder. Restructuring variability in software product lines using concept analysis of product configurations. 11th European Conference on Software Maintenance and Reengineering (CSMR'07), pages 159–170, 2007.

[LS97] Christian Lindig and Gregor Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In W. Richards Adrion, Alfonso Fuggetta, Richard N. Taylor, and Anthony I. Wasserman, editors, Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997, pages 349–359. ACM, 1997.

[LSBG21] Lukas Linsbauer, Felix Schwägerl, Thorsten Berger, and Paul Grünbacher. Concepts of variation control systems. Journal of Systems and Software, 171 :110796, 2021.

[LSSF20] Yang Li, Sandro Schulze, Helene Hvidegaard Scherrebeck, and Thomas Sorensen Fogdal. Automated extraction of domain knowledge in practice : the case of feature extraction from requirements at danfoss. In Roberto Erick Lopez-Herrejon, editor, SPLC '20 : 24th ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A, pages 4 :1–4 :11. ACM, 2020.

[Man06] Jason Xabier Mansell. Experiences and expectations regarding the introduction of systematic reuse in small- and medium-sized companies. In Timo Käkölä and Juan C. Dueñas, editors, Software Product Lines - Research Issues in Engineering and Management, pages 91–124. Springer, 2006.

[Mar91] James Martin. Rapid application development. Macmillan Publishing Co., Inc., 1991.

[Mar03] Robert Cecil Martin. Agile Software Development : Principles, Patterns, and Practices. Prentice Hall PTR, USA, 2003.

[MAZ17] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. ESPLA : A catalog of extractive SPL adoption case studies. In Maurice H. ter Beek, Walter Cazzola, Oscar Díaz, Marcello La Rosa, Roberto E. Lopez-Herrejon, Thomas Thüm, Javier Troya, Antonio Ruiz Cortés, and David Benavides, editors, Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, pages 38–41. ACM, 2017.

[MB14] Elizavita MacLennan and Jean-Paul Van Belle. Factors affecting the organizational adoption of service-oriented architecture (SOA). Inf. Syst. E Bus. Manag., 12(1) :71–100, 2014.

[MBB16] Mariem Mefteh, Nadia Bouassida, and Hanêne Ben-Abdallah. Mining feature models from functional requirements. Comput. J., 59(12) :1784–1804, 2016.

[MBB18] Mariem Mefteh, Nadia Bouassida, and Hanêne Ben-Abdallah. Feature model synthesis from language-independent functional descriptions. In Shaowen Yao, Zhi Jin, Xiaohui Cui, Bing Luo, Junfeng Wang, and Zhengtao Yu, editors, 16th IEEE International Conference on Software Engineering Research, Management and Applications, SERA 2018, Kunming, China, June 13-15, 2018, pages 151–158. IEEE Computer Society, 2018.

[McF96] Bob McFeeley. Ideal : A user's guide for software process improvement. page 237, 02 1996.

[McG08] John D. McGregor. Agile software product lines, deconstructed. J. Object Technol., 7(8) :7–19, 2008.

[MCL12] Orla McHugh, Kieran Conboy, and Michael Lang. Agile practices : The impact on trust in software project teams. IEEE Softw., 29(3) :71–76, 2012.

[MDD08] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Understanding self-organizing teams in agile software development. In 19th Australian Software Engineering Conference (ASWEC 2008), March 25-28, 2008, Perth, Australia, pages 76–85. IEEE Computer Society, 2008.

[MGMC13] Javier Miranda, Joaquín Guillén, Juan Manuel Murillo, and Carlos Canal. Assisting cloud service migration using software adaptation techniques. In 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 - July 3, 2013, pages 573–580. IEEE Computer Society, 2013.

[MOA+22] Gabriela Karoline Michelon, David Obermann, Wesley K. G. Assunção, Lukas Linsbauer, Paul Grünbacher, Stefan Fischer, Roberto E. Lopez-Herrejon, and Alexander Egyed. Evolving software system families in space and time with feature revisions. Empir. Softw. Eng., 27(5) :112, 2022.

[Mor16] Chantal Morley. Management d'un projet système d'information : principes, techniques, mise en oeuvre et outils. - 8e édition. InfoPro. Management des systèmes d'information. Dunod, 2016.

[MRA+22] Luciano Marchezan, Elder Rodrigues, Wesley K. G. Assunção, Maicon Bernardino, Fábio Paulo Basso, and João Carbonell. Software product line scoping : a systematic literature review. In Alexan-

der Felfernig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Andreas A. Falkner, Maider Azanza, Miguel Á. Rodríguez Luaces, Megha Bhushan, Laura Semini, Xavier Devroey, Cláudia Maria Lima Werner, Christoph Seidl, Viet-Man Le, and José Miguel Horcas, editors, SPLC '22 : 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A, page 256. ACM, 2022.

[MRS10] Kannan Mohan, Balasubramaniam Ramesh, and Vijayan Sugumaran. Integrating software product line engineering and agile development. IEEE Softw., 27(3) :48–55, 2010.

[MSC+13] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, Advances in Neural Information Processing Systems 26 : 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pages 3111–3119, 2013.

[MTS+17] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. Mastering Software Variability with FeatureIDE. Springer, 2017.

[Mus15] Mark A. Musen. The protégé project : a look back and a look forward. AI Matters, 1(4) :4–12, 2015.

[MZB+15] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up adoption of software product lines : a generic and extensible approach. In Douglas C. Schmidt, editor, Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pages 101–110. ACM, 2015.

[MZB+17] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up technologies for reuse : automated extractive adoption of software product lines. In Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard, editors, Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume, pages 67–70. IEEE Computer Society, 2017.

[NE08] Nan Niu and Steve M. Easterbrook. Extracting and modeling product line functional requirements. In 16th IEEE International Requirements Engineering Conference, RE 2008, 8-12 September

2008, Barcelona, Catalunya, Spain, pages 155–164. IEEE Computer Society, 2008.

[NE09] Nan Niu and Steve M. Easterbrook. Concept analysis for product line requirements. In Kevin J. Sullivan, Ana Moreira, Christa Schwanninger, and Jeff Gray, editors, Proceedings of the 8th International Conference on Aspect-Oriented Software Development, AOSD 2009, Charlottesville, Virginia, USA, March 2-6, 2009, pages 137–148. ACM, 2009.

[Nie05] Eila Niemelä. Strategies of product family architecture development. In J. Henk Obbink and Klaus Pohl, editors, Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings, volume 3714 of Lecture Notes in Computer Science, pages 186–197. Springer, 2005.

[NJ13] Linda M. Northrop and Lawrence G. Jones. Introduction to software product lines adoption. In Tomoji Kishi, Stan Jarzabek, and Stefania Gnesi, editors, 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013, page 286. ACM, 2013.

[NKSB19] Damir Nesic, Jacob Krüger, Stefan Stanciulescu, and Thorsten Berger. Principles of feature modeling. In Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo, editors, Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, pages 62–73. ACM, 2019.

[NLHS21] Hlad Nicolas, Bérénice Lemoine, Marianne Huchard, and Abdelhak-Djamel Seriai. Leveraging relational concept analysis for automated feature location in software product lines. In GPCE 2021 - 20th ACM SIGPLAN International Conference on Generative Programming : Concepts and Experiences, pages 170–183, Chicago, United States, October 2021. Association for Computing Machinery (ACM SIGPLAN).

[Nor04] Linda Northrop. Software product line adoption roadmap. Technical Report CMU/SEI-2004-TR-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2004.

[NR16] Najam Nazar and T. Rakotomahefa. Analysis of a small company for software product line adoption — an industrial case study. International Journal of Computer Theory and Engineering, 8 :313–322, 08 2016.

[NRG08] Muhammad Asim Noor, Rick Rabiser, and Paul Grünbacher. Agile product line planning : A collaborative approach and a case study. J. Syst. Softw., 81(6) :868–882, 2008.

[NWM19] Paulo Neis, Marco A. Wehrmeister, and Marcos Fonseca Mendes. Model driven software engineering of power systems applications : Literature review and trends. IEEE Access, 7 :177761–177773, 2019.

[Par76] David Lorge Parnas. On the design and development of program families. IEEE Trans. Software Eng., 2(1) :1–9, 1976.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. Software Product Line Engineering - Foundations, Principles, and Techniques. Springer, 2005.

[Pet01] Wiebke Petersen. A set-theoretical approach for the induction of inheritance hierarchies. In Lawrence S. Moss and Richard T. Oehrle, editors, Proceedings of the joint meeting of the 6th Conference on Formal Grammar (FG) and the 7th Conference on Mathematics of Language (MOL), FGMOL 2001, Helsinki, Finland, August 10-12, 2001, volume 53 of Electronic Notes in Theoretical Computer Science, pages 296–308. Elsevier, 2001.

[PGM12] Denys Poshyvanyk, Malcom Gethers, and Andrian Marcus. Concept location using formal concept analysis and information retrieval. ACM Trans. Softw. Eng. Methodol., 21(4) :23 :1–23 :34, 2012.

[PHS+08] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. Empir. Softw. Eng., 13(3) :303–337, 2008.

[PI93] Christos A. Papachristou and Venkata R. Immaneni. Vertical migration of software functions and algorithms using enhanced microsequencing. IEEE Trans. Computers, 42(1) :45–61, 1993.

[PIKD13] Jonas Poelmans, Dmitry I. Ignatov, Sergei O. Kuznetsov, and Guido Dedene. Formal concept analysis in knowledge processing : A survey on applications. Expert Syst. Appl., 40(16) :6538–6560, 2013.

[PJW14] Gaurav Pandey, Jan Jelschen, and Andreas Winter. Towards quality models in software migration. Softwaretechnik-Trends, 34(2), 2014.

[Pri06] Uta Priss. Formal concept analysis in information science. Annu. Rev. Inf. Sci. Technol., 40(1) :521–543, 2006.

[PSZ18] Konstantinos Plakidas, Daniel Schall, and Uwe Zdun. Software migration and architecture evolution with industrial platforms : A multi-case study. In Carlos E. Cuesta, David Garlan, and Jennifer Pérez, editors, Software Architecture - 12th European Conference on

Software Architecture, ECSA 2018, Madrid, Spain, September 24-28, 2018, Proceedings, volume 11048 of Lecture Notes in Computer Science, pages 336–343. Springer, 2018.

[Ram03] Juan Ramos. Using tf-idf to determine word relevance in document queries. 01 2003.

[RCC15] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. Cloned product variants : from ad-hoc to managed software product lines. Int. J. Softw. Tools Technol. Transf., 17(5) :627–646, 2015.

[RMS18] Luisa Rincón, Raúl Mazo, and Camille Salinesi. APPLIES : A framework for evaluating organization's motivation and preparation for adopting product lines. In 12th International Conference on Research Challenges in Information Science, RCIS 2018, Nantes, France, May 29-31, 2018, pages 1–12. IEEE, 2018.

[RMS19] Luisa Rincón, Raúl Mazo, and Camille Salinesi. Analyzing the convenience of adopting a product line engineering approach : an industrial qualitative evaluation. In Carlos Cetina, Oscar Díaz, Laurence Duchien, Marianne Huchard, Rick Rabiser, Camille Salinesi, Christoph Seidl, Xhevahire Tërnava, Leopoldo Teixeira, Thomas Thüm, and Tewfik Ziadi, editors, Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019, pages 73 :1–73 :8. ACM, 2019.

[Rog83] E. M Rogers. Diffusion of Innovations. The Free press, 1983.

[RPK11] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. Extraction of feature models from formal contexts. In Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Workshop Proceedings (Volume 2), page 4, 2011.

[RS11] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2), 2011.

[SA04] Outi Salo and Pekka Abrahamsson. Empirical evaluation of agile software development : The controlled case study approach. In Frank Bomarius and Hajimu Iida, editors, Product Focused Software Process Improvement, 5th International Conference, PROFES 2004, Kausai Science City, Japan, April 5-8, 2004, Proceedings, volume 3009 of Lecture Notes in Computer Science, pages 408–423. Springer, 2004.

[SB01] Ken Schwaber and Mike Beedle. Agile Software Development with Scrum. Prentice Hall PTR, USA, 2001.

[SB22] Steven She and Thorsten Berger. Formal semantics of the kconfig language. CoRR, abs/2209.04916, 2022.

[Sch97] Ken Schwaber. Scrum development process. In Jeff Sutherland, Cory Casanave, Joaquin Miller, Philip Patel, and Glenn Hollowell, editors, Business Object Design and Implementation, pages 117–134, London, 1997. Springer London.

[Sch23] Erich Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead. SIGKDD Explor., 25(1) :36–42, 2023.

[SCW12] Steven She, Krzysztof Czarnecki, and Andrzej Wąsowski. Usage scenarios for feature model synthesis. In Proceedings of the VARiability for You Workshop : Variability Modeling Made Useful for Everyone, VARY '12, page 15–20, New York, NY, USA, 2012. Association for Computing Machinery.

[SDH+12] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Generation of operational transformation rules from examples of model transformations. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings, volume 7590 of Lecture Notes in Computer Science, pages 546–561. Springer, 2012.

[SE02] Daniel Simon and Thomas Eisenbarth. Evolutionary introduction of software product lines. In Gary J. Chastek, editor, Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19-22, 2002, Proceedings, volume 2379 of Lecture Notes in Computer Science, pages 272–282. Springer, 2002.

[She13] She, Steven. Feature Model Synthesis. PhD thesis, University of Waterloo, 2013.

[SJ02] Klaus Schmid and Isabel John. Developing, validating and evolving an approach to product line benefit and risk assessment. In 28th EUROMICRO Conference 2002, 4-6 September 2002, Dortmund, Germany, pages 272–283. IEEE Computer Society, 2002.

[SJ04] Klaus Schmid and Isabel John. A customizable approach to full lifecycle variability management. Sci. Comput. Program., 53(3) :259–284, 2004.

[SLB+11] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In

Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic, editors, Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, pages 461–470. ACM, 2011.

[SM01] Gerd Stumme and Alexander Maedche. Ontology merging for federated ontologies on the semantic web. In Asunción Gómez-Pérez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing Seattle, USA, August 4-5, 2001, volume 47 of CEUR Workshop Proceedings. CEUR-WS.org, 2001.

[SMLD97] Houari A. Sahraoui, Walcélio L. Melo, Hakim Lounis, and F. Dumont. Applying concept formation methods to object identification in procedural code. In 1997 International Conference on Automated Software Engineering, ASE 1997, Lake Tahoe, CA, USA, November 2-5, 1997, pages 210–218. IEEE Computer Society, 1997.

[SP16] Adam Solinski and Kai Petersen. Prioritizing agile benefits and limitations in relation to practice usage. Softw. Qual. J., 24(2) :447–482, 2016.

[Spi05] Diomidis Spinellis. Version control systems. IEEE Softw., 22(5) :108–109, 2005.

[SR99] Michael Siff and Thomas W. Reps. Identifying modules via concept analysis. IEEE Trans. Software Eng., 25(6) :749–768, 1999.

[SRA+14] Steven She, Uwe Ryssel, Nele Andersen, Andrzej Wasowski, and Krzysztof Czarnecki. Efficient synthesis of feature models. Information & Software Technology, 56(9) :1122–1143, 2014.

[SSL08] Janice Singer, Susan Elliott Sim, and Timothy C. Lethbridge. Software engineering data collection for field studies. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 9–34. Springer, 2008.

[STB+04] Mirjam Steger, Christian Tischer, Birgit Boss, Andreas Müller, Oliver Pertler, Wolfgang Stolz, and Stefan Ferber. Introducing PLA at bosch gasoline systems : Experiences and practices. In Robert L. Nord, editor, Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, volume 3154 of Lecture Notes in Computer Science, pages 34–50. Springer, 2004.

[TC16] Sandhya Tarwani and Anuradha Chug. Agile methodologies in software maintenance : A systematic review. Informatica (Slovenia), 40(4), 2016.

[TCBE05] Thomas Tilley, Richard Cole, Peter Becker, and Peter W. Eklund. A survey of formal concept analysis support for software engineering activities. In Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors, Formal Concept Analysis, Foundations and Applications, volume 3626 of Lecture Notes in Computer Science, pages 250–271. Springer, 2005.

[TKB+14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide : An extensible framework for feature-oriented software development. Sci. Comput. Program., 79 :70–85, 2014.

[TPR+08] Marco Torchiano, Massimiliano Di Penta, Filippo Ricca, Andrea De Lucia, and Filippo Lanubile. Software migration projects in italian industry : Preliminary results from a state of the practice survey. In 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshop Proceedings (ASE Workshops 2008), 15-16 September 2008, L'Aquila, Italy, pages 35–42. IEEE, 2008.

[TPR+11] Marco Torchiano, Massimiliano Di Penta, Filippo Ricca, Andrea De Lucia, and Filippo Lanubile. Migration of information systems in the italian industry : A state of the practice survey. Inf. Softw. Technol., 53(1) :71–86, 2011.

[TTKB15] Eray Tüzün, Bedir Tekinerdogan, Mert Emin Kalender, and Semih Bilgen. Empirical evaluation of a decision support model for adopting software product line engineering. Inf. Softw. Technol., 60 :77–101, 2015.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society Series B, 63 :411–423, 02 2001.

[vdACLR19] Han van der Aa, Claudio Di Ciccio, Henrik Leopold, and Hajo A. Reijers. Extracting declarative process models from natural language. In Paolo Giorgini and Barbara Weber, editors, Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings, volume 11483 of Lecture Notes in Computer Science, pages 365–382. Springer, 2019.

[vdL02] Frank van der Linden. Software product families in europe : The esaps & café projects. IEEE Softw., 19(4) :41–49, 2002.

[vdLBK+04] Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Känsälä, and J. Henk Obbink. Software product family evaluation. In Robert L. Nord, editor, Software Product Lines, Third International

Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, volume 3154 of Lecture Notes in Computer Science, pages 110–129. Springer, 2004.

[vdLSR07] Frank van der Linden, Klaus Schmid, and Eelco Rommes. Software product lines in action - the best industrial practice in product line engineering. Springer, 2007.

[vV08] Johannes C. van Vliet. Software engineering - principles and practice (3. ed.). Wiley, 2008.

[Wei82] Heinz Weihrich. The tows matrix—a tool for situational analysis. Long Range Planning, 15(2) :54–66, 1982.

[WHH03] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In Reidar Conradi and Alf Inge Wang, editors, Empirical Methods and Studies in Software Engineering, Experiences from ESERNET, volume 2765 of Lecture Notes in Computer Science, pages 7–23. Springer, 2003.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. Experimentation in Software Engineering. Springer, 2012.

[ZHP+14] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In Yookun Cho, Sung Y. Shin, Sang-Wook Kim, Chih-Cheng Hung, and Jiman Hong, editors, Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014, pages 1064–1071. ACM, 2014.

[ZPXZ12] Gang Zhang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. Cloning practices : Why developers clone and what can be changed. In 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012, pages 285–294. IEEE Computer Society, 2012.