



HAL
open science

Interactive Prototyping of Interactions : from Throwaway Prototypes to Takeaway Prototyping

Germán Leiva

► **To cite this version:**

Germán Leiva. Interactive Prototyping of Interactions : from Throwaway Prototypes to Takeaway Prototyping. Human-Computer Interaction [cs.HC]. Université Paris Saclay (COMUE), 2018. English. NNT : 2018SACLS551 . tel-02122823

HAL Id: tel-02122823

<https://theses.hal.science/tel-02122823>

Submitted on 7 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Interactive Prototyping of Interactions: From Throwaway Prototypes to Takeaway Prototyping

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

Ecole doctorale n°580 Sciences et technologies de l'information et de la
communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 12/12/2018, par

GERMÁN LEIVA

Composition du Jury :

M. Jean-Daniel Fekete Directeur de Recherche, Inria Saclay, AVIZ	Présidente
M. Björn Hartmann Associate Professor, UC Berkeley, Department of EECS	Rapporteur
M. Nicolai Marquardt Associate Professor, UCL Interaction Centre, Department of CS	Rapporteur
Mme. Fanny Chevalier Assistant Professor, University of Toronto, Department of CS	Examineur
M. Jan Borchers Professeur, RWTH Aachen University, Media Computing Group	Examineur
M. Michel Beaudouin-Lafon PRCE, Université Paris-Saclay, ex)situ	Directeur de thèse

INTERACTIVE PROTOTYPING OF INTERACTIONS

GERMÁN LEIVA

From Throwaway Prototypes to Takeaway Prototyping

Dedicated to the loving memory of Oscar Ángel Marino.

1934 – 2004

ABSTRACT

Prototyping is essential in any design process. During the early stages, designers rely on rapid prototyping to explore ideas. Current rapid prototyping tools and techniques focus on paper representations and their disposability. However, while these throwaway prototypes are quick to create they are difficult to iterate over. I argue that rapid prototyping tools can effectively support reusable as well as throwaway artifacts for sketching interaction in early-stage design.

First, I investigate tools in the context of video prototyping. Designers experience two main barriers to use video in interaction design: the time to capture and edit the video artifacts. To aid during the capturing-phase of video prototyping I created VIDEOCLIPPER. This tool embodies an integrated iterative design method that rewards discipline but permits flexibility for video prototyping. The tool provides a storyboard-style overview for each project, with TitleCards and thumbnail images, video capture for steady-state and rough stop-motion filming, editable, reusable TitleCards, and the ability to display different paths through the video or recombine videos in new ways for redesign.

I present field studies with interaction design students using VIDEOCLIPPER in three design courses. Results suggest that participants spend less time capturing and editing in VIDEOCLIPPER than with other video tools. However, designers sometimes find it tedious to create stop-motion videos for continuous interactions and to re-shoot clips as the design evolves. Participants continuously try to reduce re-shooting by reusing backgrounds or mixing different levels of fidelity.

Inspired by this behavior, I created MONTAGE, a prototyping tool for video prototyping that lets designers progressively augment paper prototypes with digital sketches, facilitating the creation, reuse and exploration of dynamic interactions. MONTAGE uses chroma keying to decouple the prototyped interface from its context of use, letting designers reuse or change them independently. I describe how MONTAGE enhances video prototyping by combining video with digital animated sketches, encourages the exploration of different contexts of use, and supports prototyping of different interaction styles.

Second, I investigate how early designs start being implemented into interactive prototypes. Professional designers and developers often struggle when transitioning from the illustration of the design to the actual implementation of the system. In collaboration with Nolwenn Maudet, I conducted three studies that focused on the design and implementation of custom interactions to understand the mismatches between designers' and developers' processes, tools and

representations. We find that current practices induce unnecessary rework and cause discrepancies between design and implementation and we identify three recurring types of breakdowns: omitting critical details, ignoring edge cases, and disregarding technical limitations.

I propose four design principles to create tools that mitigate these problems: *Provide multiple viewpoints, maintain a single source of truth, reveal the invisible* and *support design by enaction*. We apply these principles to create ENACT, an interactive live environment for prototyping touch-based interactions. We introduce two studies to assess ENACT and to compare designer-developer collaboration with ENACT versus current tools. Results suggest that ENACT helps participants detect more edge cases, increases designers' participation and provides new opportunities for co-creation.

These three prototyping tools rely on the same underlying theoretical principles: reification, polymorphism and reuse, and information substrates. Also, the presented tools outline a new prototyping approach that I call Takeaway Prototyping. In contrast to throwaway prototypes, instead of emphasizing disposability, tools for takeaway prototyping support design by enaction and reify design artifacts to materialize the lessons learned.

RÉSUMÉ

Le prototypage est une étape essentielle du processus de design. Pendant les premières phases du processus de conception, les designers utilisent le prototypage rapide pour explorer diverses idées. Les outils et techniques actuels de prototypage se concentrent en particulier sur des représentations papier et donc destinées à être jetées. Or, alors que ces prototypes jetables peuvent être créés rapidement, ils se prêtent mal au processus d'itération. Je propose donc de créer des outils de prototypage rapide qui puissent efficacement supporter la création d'artéfacts à la fois jetables et réutilisables pour esquisser de nouvelles interactions dans les première phases du processus de design.

La première partie porte sur le prototypage vidéo. Les designers font face à deux écueils majeurs à l'utilisation de la video en design d'interaction : le temps nécessaire pour filmer et celui nécessaire pour éditer. J'ai développé VIDEOCLIPPER pour aider le processus de création de video. Cet outil intègre une méthode de design itérative qui encourage la planification et permet une vraie flexibilité pendant la création de prototypes.

Je présente les résultats de trois études sur le terrain avec des étudiants en design d'interaction. Les résultats suggèrent que les participants passent moins de temps à capturer et éditer avec VIDEOCLIPPER qu'avec les autres outils vidéos. En revanche, ils trouvent parfois difficile de créer des stop-motions pour représenter des interactions

continues et de re-filmer de nouveaux segments lorsque le design évolue.

J'ai ensuite créé MONTAGE, un outil de prototypage vidéo qui permet aux designers de progressivement augmenter leurs prototypes papier avec des maquettes numériques pour faciliter la création, la réutilisation et l'exploration d'interactions dynamiques. MONTAGE utilise l'incrustation vidéo pour découpler l'interface du prototype de son contexte d'utilisation, permettant aux designers de les réutiliser ou de les modifier indépendamment. Je décris comment MONTAGE améliore le prototypage vidéo en combinant la vidéo avec des maquettes numériques animées et encourage l'exploration d'autres contextes d'utilisation tout en permettant le prototypage de styles d'interaction différents.

La deuxième partie porte sur l'implémentation de prototypes interactifs. Les designers et développeurs professionnels ont souvent du mal à effectuer la transition de la représentation du design à son implémentation concrète. Avec N. Maudet, j'ai mené trois études sur la conception et l'implémentation d'interactions non-conventionnelles pour comprendre l'écart entre les processus, les outils et les représentations des designers et des développeurs. Nous avons découvert que les pratiques actuelles entraînent des redondances entre le travail des designers et celui des développeurs et des divergences entre le design et son implémentation. Nous identifions trois types de problèmes : l'omission de détails critiques, l'ignorance des cas extrêmes et la non prise en compte des limitations techniques.

Je propose quatre principes de design pour créer des outils qui limitent ces problèmes. Ces principes sont utilisés pour créer ENACT, un environnement interactif de prototypage d'interactions tactiles. Les résultats de deux études suggèrent que ENACT aide les participants à détecter plus de cas extrêmes, augmente la participation des designers et offre de nouvelles possibilités de co-création.

Ces trois outils de prototypage reposent sur les mêmes principes théoriques sous-jacent : réification, polymorphisme, réutilisation et substrats d'information. De même, les outils présentés mettent en œuvre une approche du prototypage que je nomme "Takeaway Prototyping" ou prototypage recyclable. Par contraste avec les prototypes jetables, les outils pour le prototypage recyclable permettent le design par énonciation et réifient des artefacts de conception pour matérialiser la progression du design.

PUBLICATIONS

Germán Leiva¹, Nolwenn Maudet¹, Wendy E. Mackay, and Michel Beaudouin-Lafon (2018). “Enact: Reducing Designer-Developer Breakdowns when Prototyping Custom Interactions.” In: *ACM Transactions on Computer-Human Interaction (TOCHI)*.

DOI: [10.1145/3310276](https://doi.org/10.1145/3310276)

Germán Leiva, Nolwenn Maudet, and Michel Beaudouin-Lafon (2018). “Towards Collaborative Prototyping Tools for Interaction Design.” In: *Workshop on Digital Tools in Collaborative Creative Work at NordiCHI 2018*.

Germán Leiva and Michel Beaudouin-Lafon (2018). “Montage: A Video Prototyping System to Reduce Re-Shooting and Increase Re-Usability.” In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18. Berlin, Germany: ACM Press*.

DOI: [10.1145/3242587.3242613](https://doi.org/10.1145/3242587.3242613)

Nolwenn Maudet¹, **Germán Leiva**¹, Michel Beaudouin-Lafon, and Wendy E. Mackay (2017). “Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems.” In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing - CSCW '17*.

DOI: [10.1145/2998181.2998190](https://doi.org/10.1145/2998181.2998190)

Carla Griggio, Nam Giang, **Germán Leiva**, and Wendy Mackay (2016). “The UIST video browser: Creating shareable playlists of video previews.” In: *UIST 2016 Adjunct - Proceedings of the 29th Annual Symposium on User Interface Software and Technology*.

DOI: [10.1145/2984751.2985703](https://doi.org/10.1145/2984751.2985703)

¹ The first two authors contributed equally to this work.

ACKNOWLEDGMENTS

This dissertation and my research were possible thanks to the support of many people.

I want to thank my supervisor Michel Beaudouin-Lafon for believing in me at the beginning of this journey, giving me freedom but also a lot of support to work on a topic so dear to my heart. Thank you very much for sharing your experience, having interesting discussions and helping me in this *sometimes* challenging journey. I also want to give a huge thanks to Wendy Mackay. I learned so much from you; thank you for sharing your vast knowledge about conducting research and all your experience. I will always be grateful to both of you for bringing Human-Computer Interaction into my life.

I would also like to thank the members of my jury: Jean-Daniel Fekete, Björn Hartmann, Nicolai Marquardt, Fanny Chevalier, and Jan Borchers. Thanks to the reviewers for their insightful reports and all the jury members for the helpful questions during the presentation. I hope to see you all at the upcoming conferences.

I also want to thank many people from the ex)situ group and friends. First, this dissertation would not have been possible without the collaboration that I had with Nolwenn Maudet. We started working together in 2015, studying designer-developer collaboration for at least four years. Let's say that Nolwenn was the voice of the designer and I was the voice of the developer. I think both of us got a deeper understanding of these communities, not only from our observations but also due to our collaboration. Thank you very much for being a fantastic person to work with and a great friend, I hope to see you soon in Tokyo.

Marianela Ciolfi, thanks for your infinite altruism, for those essentials breaks and chitchats but also your invaluable feedback and comments every time that I needed them. Jessalyn Alvina thanks for the long talks, sometimes about research, sometimes about life. You were both great office-mates.

Ignacio Avellino, we missed you in the lab. I am happy to have met you. Thank you very much for your kindness and your friendship. Hanging around with you in little Argentina was terrific.

To the "third-year students": Michael Wessely and Abby Wanyu Liu. It was great to have those mini holidays in Berlin, even if I was running with the thesis. We did it.

Thank you Yujiro Okuya for sharing these three years with me and being our Japan connection.

Thanks to the "old members" of the lab: Jean-Baptiste Louvet, Alex Zinenko, Ghita Jalal, and many more. Thanks to the new members –at

least from my perspective— Philip Tchernavskij, Stacy Hsueh and Jean-Philippe Rivière. For the post-doc view of the world: John MacCallum and Andrew Webb.

Thanks to Lai Linghua for being my first co-supervised master student in the topic of video prototyping. Thank you for conducting the studies that made the basis for [Section 3.2](#).

Thanks to Jiannan Li and Viktor Gustafsson for being so cool and giving me feedback on earlier versions of the Enact paper. Thank you, Viktor, for taking care of the iPads.

To all the interns, master students and collaborators: Midas Nouwens, Téo Sanchez, Brennan Jones, Cheng Cheng Qu, Diana Lipcanu, Nam Giang, Francesco Vitale, Jingyi Li, Jiali Liu, Alexander Eiselmayer, Krishnan Chandran, and many more. Thanks to all the people that helped with Montage: Miguel Renom, Tong Xue, Dimitrios Christaras and Pierre Dragicevic.

Thanks Alexandra Merlin for helping me navigate the bureaucratic waters of France, it would have been impossible without your help. Thank you to the permanents: Theophanis Tsandilas, Cédric Fleury, Sarah Fdili Alaoui, Baptiste Caramiaux. Thank you Joanna McGrenere for your insightful comments and your positive energy. Nicolas Taffin, you are a remarkable human being, thank you for the book.

Thanks to the people from other labs and organizations that in one way or another helped me: Caroline Appert, Anastasia Bezerianos, Evanthia Dimara, Arnaud Prouzeau, Rafael Morales Gonzalez, María-Jesús Lobo, Hugo Romat, Bruno Le Dantec, and many more.

Thanks to all the students, designers, and developers that shared their stories and used our tools.

I would also like to thank the STIC Doctoral School, Université Paris-Saclay, Université Paris-Sud and Inria for supporting this research.

I want to thank all my friends from Argentina; they were always there for me even if I was so far away. I want to thank my family for their unconditional affection and support.

Finally and most importantly, I would like to thank the love of my life. Carla Griggio, *mi Líder*. Thank you for so much. We survived.

CONTENTS

1	INTRODUCTION	1
1.1	Thesis Contributions	2
1.2	Outline	3
2	BACKGROUND: PROTOTYPING IN INTERACTION DESIGN	5
2.1	A Brief History of Design and Prototyping	5
2.2	Software Process Models	9
2.2.1	Waterfall model	9
2.2.2	Agile Methodologies	12
2.2.3	Participatory Design	12
2.3	Prototyping and Prototypes	14
2.3.1	Prototyping as a process	14
2.3.2	Prototypes as artifacts	18
2.3.3	Low vs High Fidelity Prototypes	20
2.3.4	Prototyping Dimensions	22
I	VIDEO PROTOTYPING	
3	WHY VIDEO PROTOTYPING?	27
3.1	Related Work: Early-stage Prototyping Tools and Techniques	28
3.1.1	Commercial tools	29
3.1.2	Video-based Tools	30
3.1.3	Video Prototyping Systems	42
3.2	How do interaction designers use video today?	44
3.2.1	Method	44
3.2.2	Results and Discussion	45
4	VIDEOCLIPPER: PLANNING VIDEO CAPTURE ON AN INTERACTIVE STORYBOARD	51
4.1	A Typical Video Prototyping Session	51
4.2	Design Goals	56
4.3	Video Prototyping with VideoClipper	57
4.3.1	Implementation	61
4.4	Field studies	62
4.4.1	First contact with real users	62
4.4.2	Comparing mobile iMovie with VideoClipper	64
4.4.3	Using VideoClipper in a longer course	66
4.5	Limitations and Future Work	68
5	MONTAGE: COMBINING VIDEO PROTOTYPES TO PROVIDE REUSE	71
5.1	Decoupling context and interface with Montage	71
5.2	Limitations when combining paper and video	72
5.3	MONTAGE Mirror: Prototyping continuous feedback	74
5.4	MONTAGE Chroma: Reusing captured interactions	76

5.4.1	Exploring design alternatives and multiple contexts of use	77
5.4.2	Supporting multiple interaction styles	78
5.5	Implementation	80
5.6	Limitations and Future Work	81
5.7	Conclusion	82
II INTERACTIVE PROTOTYPING		
6	WHY DESIGNER-DEVELOPER COLLABORATION?	85
6.1	Motivation and Methodology	87
6.1.1	Methods standardization	88
6.1.2	Boundary objects	89
6.1.3	Our approach	89
6.2	Related Work: Designer-Developer Practices and Tools	90
6.2.1	Studies of designer-developer practices	90
6.2.2	Prototyping tools	91
6.2.3	Summary	93
7	STUDYING DESIGNER-DEVELOPER COLLABORATION	95
7.1	Study One: Understanding Designer-Developer Collaboration Practices	95
7.1.1	Method	95
7.1.2	Results and Discussion	96
7.1.3	Summary	104
7.2	Study Two: Analyzing Designer-Developer Breakdowns	104
7.2.1	Method	104
7.2.2	Results and Discussion	105
7.2.3	Summary	107
7.3	Study Three: Exploring Design Solutions	107
7.3.1	Results and Discussion	109
7.3.2	Summary	112
8	A PRINCIPLED PROTOTYPING TOOL	113
8.1	Design Principles for Designer-Developer Collaborative Tools	113
8.1.1	Principle One: Provide multiple viewpoints	114
8.1.2	Principle Two: Maintain a single source of truth	115
8.1.3	Principle Three: Reveal the invisible	116
8.1.4	Principle Four: Support design by enaction	116
8.1.5	Summary	118
8.2	Enact: A Tool for Collaborative Prototyping of Touch-based Interactions	119
8.2.1	Overview	119
8.2.2	Use scenario	120
8.2.3	Drawing the user interface	122
8.2.4	Providing concrete input examples	123
8.2.5	Generating an animation from the storyboard	124
8.2.6	Programming interaction	124

8.2.7	Target device and mirror	129
8.2.8	Assisted testing	129
8.2.9	System Implementation	130
8.2.10	Limitations and Future Work	131
8.3	Study Four: Assessing ENACT	132
8.3.1	Method	133
8.3.2	Results and Discussion	134
8.4	Study Five: Comparing ENACT with Traditional Tools	135
8.4.1	Method	135
8.4.2	Results and Discussion	138
8.5	Conclusion	144
9	FROM THROWAWAY PROTOTYPES TO TAKEAWAY PROTOTYPING	147
9.1	Reification, Polymorphism, and Reuse	147
9.1.1	Reification	147
9.1.2	Polymorphism	148
9.1.3	Reuse	148
9.2	Information Substrates	149
9.3	Takeaway Prototyping	151
9.3.1	Design by enaction	152
9.3.2	Reuse design artifacts	152
9.3.3	Supporting Takeaway Prototyping	153
10	CONCLUSIONS AND FUTURE WORK	157
10.1	Implications for Future Research	158
A	APPENDIX	161
A.1	Survey about Video in Interaction Design	161
A.2	Questionnaires VIDEOCLIPPER	168
	BIBLIOGRAPHY	171

LIST OF FIGURES

- Figure 2.1 The Levallois technique of flint-knapping by José-Manuel Benito Álvarez. Licensed under . [6](#)
- Figure 2.2 Parthenon by jhadow. Licensed under . [7](#)
- Figure 2.3 The revolutionary cast iron cooking pot. Copyright Ironbridge Gorge Museum Trust. [7](#)
- Figure 2.4 1914 Ford Model T. Four Cylinder, Twenty Horsepower by LibertyGroup25. Licensed under . [8](#)
- Figure 2.5 The SAGE system. Information is sent to the *direction center* (in the middle) from multiple sources. At the top, from left to right: long-range radars, airborne early-warning planes, radar picket ships, gap-filler radars, offshore air defense radars, air traffic control information. At the bottom, instructions can be sent to missiles and aircraft after processing the information. Image from the Computer History Museum (*In Your Defense*). [9](#)
- Figure 2.6 The waterfall process used during the construction of the SAGE system. Image reproduced from Benington (1983). [10](#)
- Figure 2.7 Incremental vs Iterative. Copyright: Henrik Kniberg <https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp> [11](#)
- Figure 2.8 The Waterfall Model has long development cycles (analysis, design, implementation, test) while the iterative methods, such as the Spiral Model, mix these activities in a shorter development cycle. XP blends all the activities, a little at a time, during the entire software development process. Figure based on the model presented in Beck (1999) [13](#)
- Figure 2.9 Calculator from Hertzfeld (1982) [14](#)

- Figure 2.10 Horizontal and Vertical prototyping. Horizontal prototyping covers multiple features but disregards depth of the functionality. On the other hand, vertical prototyping provides full depth of functionality of a minimal number of features. Figure reproduced from Nielsen (1994) 16
- Figure 2.11 The system development life-cycle in mechanical engineering compared with information system engineering. Figure from Janson and Smith (1985) 18
- Figure 2.12 What do prototypes prototype. Figure from Houde and Hill (1997) 20
- Figure 2.13 Table based on the dimensions presented by Beaudouin-Lafon and Mackay (2003) for analyzing prototypes: evolution, representation, precision, and interactivity. Precision is subdivided into the three interaction design dimensions presented by Cooper (1995): content, form and behavior. A range of precision covers one or more of the following levels: **Low**, **Medium** and **High**. For example, **LH** denotes a range from low to high, i.e. including medium, while **L** is only low. Plus (+) and minus (-) signs highlight small variations on the precision for that particular technique, i.e. higher or lower respectively. 24
- Figure 3.1 Commercial prototyping tools are ill-equipped to support rapid prototyping and highly dynamic interactions. I want to increase the speed of video prototyping while at the same time increasing its power to prototype custom and dynamic behaviors. 30
- Figure 3.2 Simplified version of the design framework presented by Mackay, Ratzer, and Janecek (2000). The framework presents two main domains on the vertical axis: technology and use. The horizontal axis illustrates that design activities work on the abstractions and the details of each of these domains. Each activity has a video artifact as input and as an output. The four main considerations in the design framework are interaction techniques, design principles, context of use and interaction patterns. 31

- Figure 3.3 *Storyboards* help designers to organize the sequence of events before recording a video prototype. Here, each panel represents the computer interface. The top-left corner illustrates the user actions, i.e. mouse inputs. The text below each panel explains the user inputs and the system responses. Figure from Vertelney (1989). 34
- Figure 3.4 With *animated drawings*, a background is copied and reused. Sketches drawn on top of each background represent the look of the interface at short time intervals. When successive images are filmed at 24 frames per second they create the illusion of motion. Figure from Vertelney (1989). 35
- Figure 3.5 By using *cutouts* to represent movable objects, designers avoid laborious stop-motion. The cutouts represent interface elements, such as menus, or annotations in the story, such as speech bubbles. Figure from Vertelney (1989). 36
- Figure 3.6 *Physical models* can represent 3D interactions. The same stop-motion techniques applied to 2D also work for 3D prototyping. The camera can even move around the physical space to prototype navigation alternatives. Figure from Vertelney (1989). 37
- Figure 3.7 A video prototyping of a professor interacting with a virtual agent. The professor requested the agent to copy information about "deforestation in the Amazon" in the physical card-storage device. From "The Knowledge Navigator" by Apple Inc. 39
- Figure 3.8 The prototype of an interactive curved desktop display. Placing objects on the flat area allows the system to scan the contact area, e.g., to scan a document. The workstation can be controlled with a mouse or via voice commands. From "The Starfire" by Sun Microsystem Inc. 40
- Figure 3.9 The Virtual Studio at CAVI (Centre for Advanced Visualization and Interaction) in Aarhus, Denmark. 41
- Figure 3.10 The miniStudio system uses projections to prototype proxemic interaction. Figure from Kim, Kim, and Nam (2016). 43

- Figure 3.11 Distribution of the participants' design experience. Half had 1-3 years of experience. The other half: 29% less than one year, 15% between 4-8 years and 6% between 9-12 years. The average experience is 6.8 years. 45
- Figure 3.12 Amount of design artifacts from the most recent project. Hand-drawn and computer-drawn sketches are among the most created artifacts. The least created are video illustrations and computer animations accompanied by a mix of tools, such as software mockups –Android and HTML5–, prototyping tools –InVision–, and presentation software –Google Slides–. 46
- Figure 3.13 How much time was spent creating these design artifact? For 59% of the participants, the creation of hand-drawn sketches only took between 0-20% of the process. Most participants said that computer-drawn sketches took between 0-60%. However, one participant responded that creating the computer-drawn artifacts took between 80%-100% of the time. Another participant reported that video and computer animations took between 60%-80% of the process. 47
- Figure 3.14 We asked participants about their use of video as a design (Figure 3.14a), communication (Figure 3.14b) and documentation tool (Figure 3.14c). 48
- Figure 3.15 The main barriers to using video are the lack of time or resources to edit (76%) and the lack of time or resources to prepare for and record video (68%). 49
- Figure 4.1 A digital User Interface (UI) storyboard created in Apple Xcode, focused on the user interface layout and the flow between screens. 52
- Figure 4.2 A narrative hand-drawn storyboard that combines sketches with textual descriptions and sticky notes. 53
- Figure 4.3 Designers improvise a background of the Eiffel Tower by sketching over a whiteboard. 54
- Figure 4.4 The designers use transparent paper to mimic the dragging feedback on top of the paper prototype. 55

- Figure 4.5 The VIDEOCLIPPER's storyboard view organizes each video prototype as collection of ordered Lines. Each Line starts with a TitleCard followed by the corresponding video clips. Designers can open the capture view by pressing the camera icon on the right. 57
- Figure 4.6 VIDEOCLIPPER TitleCard's editor. The duration of the TitleCard can be edited with the button on the lower right corner. 58
- Figure 4.7 VIDEOCLIPPER's capture view. On the left, a column of TitleCards. On the top, the video clips of the selected Line. In the center, a preview of the camera, in this case with an active ghost image. 59
- Figure 4.8 Adding a 'ghost' of the last recorded frame lets designers align the shots to create rough "invisible cuts". 60
- Figure 4.9 Moving Lines in VIDEOCLIPPER. 61
- Figure 4.10 The VideoClipper data model. 62
- Figure 4.11 Reported average times in minutes collected from 16 students, the error bars represent the standard deviation. The average capturing time was 3h28m for iMovie and 2h17m for VIDEOCLIPPER. The average editing time was 1h8m for iMovie and 42m for VIDEOCLIPPER. 64
- Figure 4.12 Reported times collected from 41 students using VIDEOCLIPPER in a longer course. The average capturing time was 3 hours and the average editing time was 47 minutes. 67
- Figure 4.13 A video prototype of a touch-based drawing application with vertex recognition. 68
- Figure 4.14 A video prototype of a draggable and resizable lens. 69
- Figure 4.15 Students mixed different fidelities to illustrated multiple aspects of the design. 69
- Figure 4.16 A video prototype of an interaction with a real plant. 70
- Figure 4.17 A student presenting on top of the video prototype playing in the background. 70

- Figure 5.1 An overview of the MONTAGE video prototyping system. Here, two designers, a wizard and a user-actor, collaborate side-by-side. The *UserCam* captures the user-actor enacting user inputs in the actual context, i.e. using his phone at the office. The *WizardCam* captures the paper prototype and the *Canvas* captures the wizard's digital sketches. 72
- Figure 5.2 OctoPocus with traditional video prototyping. The designers create a rough stop-motion movie with only four stages of the interface, resulting in a poor representation of the dynamic interaction. 73
- Figure 5.3 Mirror mode: the *WizardCam* live streams to both, the *Canvas* (a) and to the prototyped device (b) in the *context*. The *UserCam* only streams to the *Canvas* (c). Finally, the *Canvas* sends the sketches to the prototyped device to complete the mirroring of the *interface* 75
- Figure 5.4 Chroma mode: the *UserCam* captures the *context* (a) and the *WizardCam* captures the *paper prototype* (b); Both live-stream video to the *Canvas*. Designers draw digital sketches over the streamed *paper prototype* to represent the *interface* (c). In the *Canvas*, the green screen is replaced with a perspective transformation of the *interface* to create the *final composition* (d). 77
- Figure 5.5 The *Canvas* sketching interface: The *final composition* (left) and the *interface* (right) show the "user overlay". Both sides have a list of sketches and animation controls at the bottom. The in/out buttons make the selected sketches appear/disappear. The sliders control the stroke-start, now at 0%, and the stroke-end, now at 100%. 78
- Figure 5.6 Green screens let designers explore the same interface in multiple contexts, e.g., stationary on a desk or walking with a phone or a watch. 79
- Figure 5.7 Sketches over the context can represent Augmented Reality (AR) visual elements. 80

- Figure 6.1 An example of a custom interaction in the Paper mobile app from FiftyThree. In the first screen, the user has selected the scissor tool and draws a circular area with one finger. In the second screen, the user drags this area to move the content to a new position and taps outside the circular area with another finger to create a copy of the selected area. In the third screen, the user drags the selected area to reveal the copied shape. 86
- Figure 7.1 A StoryPortrait has a summary title, a photograph of a key artifact, and a top-to-bottom timeline to show the successive steps of the collaboration between designers (on the left) and developers (on the right). Participant's quotes and drawings enrich the story. Here, P10_{dv} started his story by saying that "the motion designer sends us a video". The designer sent the mockup-up, user journey and video file via email. P10_{dv} translated the video into code: "[I] try to find the useful information" in the provided artifacts. He failed to extract the relevant information about the animation, such as the keyframe timestamps and the interpolation curves. P10_{dv} finally asked the designer for a text file with all the animation parameters. The designer sent by email a the text file created by hand with all the details of the animation. Creating this file was "time-consuming and boring to produce". P10_{dv} finished his story by saying: "we try to set up standards, but we have not found the right one so far". 99
- Figure 7.2 Key design breakdowns between designers and developers: *Missing information*: Designers do not communicate necessary details. *Edge Cases*: Designers do not consider certain problematic situations. *Technical constraints*: Designers are not aware of technical limitations. 101
- Figure 7.3 Relationship between interaction type (standard vs. custom) and developer involvement. Lack of developer involvement in the early phase of custom interaction design is correlated with problematic or impossible implementation. Nomenclature: P1.a identifies the first story of participant one, while P1.b identifies the second story of participant one. 103

- Figure 7.4 First meeting. The developer interacts with an existing application to discuss possible interactions with two designers while another represents it on paper. 106
- Figure 7.5 The *pinch-to-create* interaction is based on the Clear to-do list mobile app. First the user puts down two fingers simultaneously. Then, by spreading them, the new item is revealed progressively. Finally, the new item is created when the user lifts her fingers off the screen. 109
- Figure 7.6 An example of a designer's representation of *pinch-to-create*. The designer depicts a continuous interaction by discretizing key visual states and adding user input annotations. 110
- Figure 7.7 (a) A designer drew a snapshot of the lasso-fill interaction at four points in time. (b) A developer created a diagram connecting primitive graphical elements and functions with user inputs. (c) The designer merged the two representations with an example. 111
- Figure 8.1 ENACT uses a target mobile device and a desktop interface with five areas: a storyboard with consecutive screens, an event timeline with a handle for each screen, a state machine, a code editor and a device mirror 119
- Figure 8.2 Scenario. The first screen of the storyboard shows the initial look of the interaction. The two items created in the first screen (light blue) propagate to the second screen, where a third item (in yellow) is added (a). Each screen has a corresponding handle in the input timeline (b). The state machine shows the selected transition in green and the active state in red (c). State machine actions can be edited in the code editor (d), either by coding or by linking a storyboard elements to create *design references*. In the third screen of the storyboard, a measure M1 is added between the top and bottom rectangle (e). 121

- Figure 8.3 Recording an input on the target device. The input events are saved in the timeline (a), the designer can navigate the recorded inputs by dragging them or by moving the associated screen handle in the timeline (b). Current touches are displayed as translucent grey ellipses on each screen, the radius of the ellipse represents the size of the touch (c). [123](#)
- Figure 8.4 A state machine for a pinch interaction. The state machine diagram is interactive: states and transitions can be added, removed or edited. The selected state, in green, can be edited in the code editor. The currently active state and transition are highlighted in red. [125](#)
- Figure 8.5 The editor shows the code of the selected state or transition. Here, a touchmove transition is selected. The guard ensures that the transition triggers only when touch T0 is inside rectangle R1. In the action, the developer has defined a mapping between the positions of the touch and the rectangle. The editor creates *design references* around recognized design elements: `$.T0.position`, `$.R1.position` and `$.S1.R1.position.y`. The number 237 is a *local design reference* representing the y-position of rectangle R1 in screen S1. If the storyboard changes, this value will also change. [126](#)
- Figure 8.6 The designer created two measures, one between the touches and the other between the two rectangles. These measures are invisible on the target device (left) but they are revealed in the device mirror (right) and updated in real time to facilitate debugging. [128](#)
- Figure 8.7 Testing in ENACT combines the code with the recorded user inputs. To run the test, the designer presses the Test button (a). Here, the first and second screen match the test result and their screen handles are shown in green (b). The third screen has a mismatch displayed as a dashed orange line and the screen handle is shown in yellow (c). [130](#)

- Figure 8.8 Visual specification of rules in the earlier version of ENACT tested in Study Four. Input-output rules connect one input with one or more outputs. The first column shows the identifier of the element (T_0 and R_1), the second and third columns refer to the X and Y-axis respectively. This rule maps the touch's Y-axis translation (\Downarrow) to rectangle's Y-axis scale (\Uparrow) with a ratio of 0.5. Each axis transformation can have a minimum and maximum value taken from the storyboard. 133
- Figure 8.9 Study Five. Each pair consist of one designer (P_{ds}) and one developer (P_{dv}). The designer has 10 minutes to create a design artifact for the initial version of the proposed interaction, then the developer has 15 minutes to implement it. Finally, both work together for 15 minutes to evaluate the current implementation and work on the final version of the interaction. 136
- Figure 8.10 P_{4ds} and P_{4dv} working side-by-side on the final version of *pan-and-stamp*. On the left, the developer performs an off-device mimicking gesture with his left hand to understand the proposed design. On the right, the designer performs an on-device mimicking gesture with both hands to communicate the design. 138
- Figure 8.11 Number of on-device mimicking gestures, either on the mobile device or the emulator provided by the IDE, per participant during the collaborative side-by-side phase. Designers and developers performed significantly more interactions on the device with ENACT than with TRADITIONAL tools. This suggests that with ENACT, designers participate more during the side-by-side phase and developers perform more contextualized actions on the real device during implementation. 141
- Figure 9.1 Extended version of the table presented in Figure 2.13. ENACT embodies most of the principles of Takeaway Prototyping in order to provide rapid as well as iterative prototypes. On the other hand, VIDEOCLIPPER (Video Prototyping) and MONTAGE would need more mechanisms to iterate the prototype to better support Takeaway Prototyping. 154

ACRONYMS

UI	User Interface
GUI	Graphical User Interface
WOz	Wizard of Oz
WIMP	Windows, Icons, Menus, Pointer
IDE	Integrated Development Environment
AR	Augmented Reality
DRY	Don't Repeat Yourself
MVC	Model-view-controller

INTRODUCTION

Prototyping is an essential activity to support the exploration and communication of interactive systems (Beaudouin-Lafon and Mackay, 2003). During early-stage prototyping, designers rely on rapid prototyping to create throwaway prototypes that are discarded when they have served their purpose. Throwaway prototypes usually take the form of paper prototypes made with art supplies such as pen-and-paper, markers and sticky notes. These low-tech representations are inexpensive and allow the participation of non-experts (Bødker and Grønbaek, 1991). However, these static paper representations poorly support highly customized dynamic interactions (Bailey and Konstan, 2003), such as gesture-based interfaces or AR. Designers need more time-consuming computer-based tools to prototype in these interaction styles, thus hindering the participation of non-experts. How can designers prototype highly dynamic interactions without programming and with the speed of rapid prototyping techniques?

Another shortcoming of current static design artifacts is their lack of support for an incremental and iterative process. Initial paper representations, such as hand-drawn sketches, are quick to create but force designers to redo their designs when exploring variations of previously sketched ideas. Also, interaction designers rely on other design artifacts such as mockups, wireframes and specifications (Newman and Landay, 2000). Modifying one artifact requires designers to replicate the changes in all the others to preserve their consistency. Even more, once an early implementation starts, developers might need to replicate the whole work with their own tools. What is the effect of this redundancy among design artifacts? How can we reduce this repetitive work within and across designer-developer activities?

First, I decided to explore these questions in the context of *video prototyping*. Video has been long identified as a powerful medium for prototyping interactions (Mackay, 1988). Existing video tools support either the capture or post-production phases (Berthouzoz, Li, and Agrawala, 2012; Girgensohn et al., 2000). However, we still lack tools to support the prototyping process, not just the prototyping product, i.e. improving the quality of the resulting video. Designers can use video prototypes to explore alternatives, *feel* the interaction while acting it out and communicate interaction in context (Mackay, Ratzer, and Janecek, 2000). However, video is not as widely used in early-stage design as other prototyping media (Carter and Hundhausen, 2010). Do designers find video useful for interaction design? What are the

barriers for using video as a rapid prototyping technique? How can we design tools that facilitate the process of video prototyping?

Sometimes just illustrating the interaction with paper or video prototyping is not enough, and designers need to collaborate with developers to create fully *interactive prototypes*. However, the collaboration between these communities of practice is not smooth (Moffett, 2014). Designers and developers of interactive systems have different backgrounds and skills (Buxton, 2007) and focus on different aspects of the design process (Löwgren, 1995). Despite these differences, they need to collaborate to create interactive systems. The first step to improve this situation is to understand the causes of the existing problems to represent, communicate and interpret interaction designs. Do designers and developers have difficulties reusing previous work? Does it have an impact on the early implementation of interactions?

Commercial prototyping tools focus on graphical authoring of static images and offer poor support for dynamic behaviors. These interactions are constrained to a small predefined set of standard inputs and animated transitions. To prototype more complex behaviors, designers need to "code" using textual representations, visual languages or parameters hidden behind intricate menus. How can tools provide a shared sandbox to support these two different disciplines?

THESIS STATEMENT I argue that rapid prototyping tools can effectively support reusable as well as throwaway artifacts for sketching interaction in early-stage design.

1.1 THESIS CONTRIBUTIONS

This dissertation provides empirical findings from multiple studies, technical contributions in the form of fully functional prototyping tools, and theoretical contributions that introduce design guidelines and a new prototyping approach.

EMPIRICAL CONTRIBUTIONS: I found that

- designers' main barriers to using video for interaction design are the laborious activities of capturing and editing;
- designers rely on multiple artifacts to communicate interaction to developers with redundant information;
- the translation from interaction design to software development requires extensive rework and usually creates misalignments between design and implementation; and
- designer-developer collaboration during the creation of interactive systems suffers from recurring breakdowns.

TECHNICAL CONTRIBUTIONS: I designed and implemented

- VIDEOCLIPPER, a tablet-based video prototyping tool integrating planning, recording and lightweight editing with an interactive storyboard;
- MONTAGE, a video prototyping system that combines paper prototyping and digital sketches with Wizard of Oz (WOz) and chroma key compositing; and
- ENACT, a principled collaborative prototyping tool that provides multiple and integrated representations of the interaction under construction.

THEORETICAL CONTRIBUTIONS: I derived

- design principles for the creation of collaborative interaction prototyping tools aimed at reducing designer-developer breakdowns; and
- *Takeaway Prototyping*, a new prototyping approach that materializes the lessons learned among iterations.

1.2 OUTLINE

This dissertation is divided into two parts: video prototyping and interactive prototyping.

CHAPTER 2: I present design and prototyping from a historical perspective. I review different methodologies for the construction of interactive systems and the predominant role of prototyping in all of them. I introduce classifications of prototyping, as a process, and of prototypes, as products.

CHAPTER 3: At the beginning of Part 1, I present related work on video-based prototyping tools. I explore the main barriers to using video in interaction design. One barrier is the time-consuming activity of preparing for and capturing video, and the other is editing.

CHAPTER 4: I introduce VIDEOCLIPPER as video prototyping tool to aid in planning and recording. VIDEOCLIPPER provides an interactive storyboard featuring many reusable elements focused on capturing on-the-go. I report on four studies with VIDEOCLIPPER in multiple design courses.

CHAPTER 5: I introduce MONTAGE as a video prototyping system to reduce re-shooting and increase reuse. MONTAGE re-purposes mobile devices as remote cameras to provide a mobile green screen studio focusing on combining and reusing video prototypes. I illustrate many uses cases of MONTAGE and how it supports the prototyping of diverse interaction styles.

CHAPTER 6: At the start of Part 2, I present the motivation for studying designer-developer collaboration. I review related work on collaboration and prototyping tools for interactive prototyping.

CHAPTER 7: I present three studies to understand current designer-developer problems. The first study focuses on their previous projects; the second, on direct observation of participants collaborating; and the third, on envisioning potential solutions for the identified problems.

CHAPTER 8: Based on the findings in Chapter 7, I introduce four principles for the creation of interactive prototyping tools to mitigate designer-developer breakdowns. I apply these principles to create a new prototyping tool. ENACT features many reusable artifacts that support both designer and developer representations. I report on two studies to assess ENACT in both individual and collaborative use.

CHAPTER 9: I analyze the work presented in the previous chapters using the theoretical principles of Reification, polymorphism and reuse, and the concept of information substrates. Supported by these principles, I present a new prototyping approach called Takeaway Prototyping.

CHAPTER 10: I conclude the dissertation with a summary of the main contributions and directions for future research.

BACKGROUND: PROTOTYPING IN INTERACTION DESIGN

This dissertation investigates how prototyping tools can support reuse in early-stage design. However, there are different views on what is prototyping and what is meant by "early" stages. This chapter introduces these concepts through a brief historical review.

2.1 A BRIEF HISTORY OF DESIGN AND PROTOTYPING

Human history is tightly linked to the creation of artifacts. Some of the oldest stone tools found were dated about 3.33 million years ago (Harmand et al., 2015). The word "design" appeared first as a verb around the 14th century and later as a noun around 1580 (Online Etymology Dictionary, 2018a). Design as a verb comes from the Latin *designare* "mark out, point out; devise; choose", from *de* "out" combined with *signare* "to mark," from *signum* "identifying mark, sign". To design can be seen as the action of choosing, making decisions over anything that has an impact on the construction of an object. On the other hand, design as a noun comes from the Middle French word *desseign*, i.e. "a scheme or plan in the mind" (Online Etymology Dictionary, 2018b). In this view, design is deciding what to build, i.e. the form, and how to build it, i.e. the process. Even though early humans did not have a word for it, they designed different stone tools, such as hammerstones and hand-axes (Figure 2.1). However, these artisans created artifacts with no clear divide between the design and the construction of the object (Sparke, 2009). In other words, there was no explicit materialization of the design, e.g., a specification or a plan, beyond the created artifact. There is evidence, however, that they trained younger ones to master the art of creating these tools.

The word prototype comes from the Greek *protos* "first" and *typos* "impression, mold, pattern" (Online Etymology Dictionary, 2018b). From a reductionist point of view, a prototype is "a first impression" or "a primitive form", and thus, prototyping is the action of building these early object samples. If we accept for a moment this broad definition, prototyping has also always existed alongside humans (Guggenheim, 2010). However, to see the roots of modern product design and more sophisticated uses of prototypes, we need to move ahead in time to classical antiquity (Bürdek, 2005).

Some historians believe that the Parthenon (Figure 2.2) and other buildings of the same era were based on miniature wooden prototypes or pre-existing buildings (The American Society of Mechanical

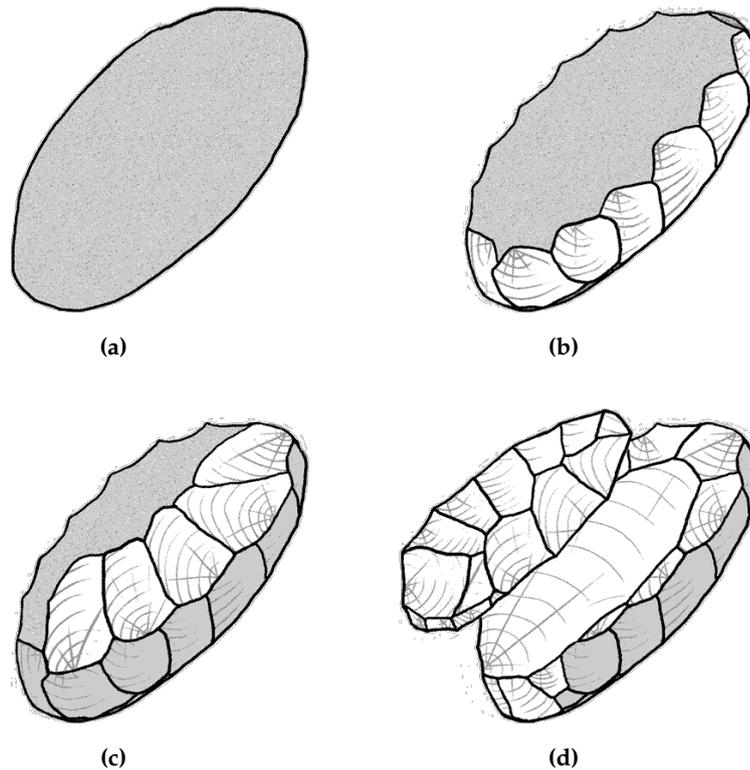


Figure 2.1: The Levallois technique of flint-knapping by José-Manuel Benito Álvarez. Licensed under .

Engineering, 2018). Following this view, prototypes are more than an early version of the final object; they can also be a materialization of the design, a model to gain inspiration or to study ideas.

Prototypes helped ancient designers make informed decisions before committing to a particular design. One of the oldest documents talking about design comes from a Roman architect, artist, and military engineer called Vitruvius (c. 80-10 BC). In his multi-volume book *De architectura*, Vitruvius talks about the multidisciplinary mindset of the architect: "an architect has to be interested in art and science, as well as being versed in rhetoric and having a good knowledge of history and philosophy" (Rowland, Howe, and Others, 2001). Vitruvius' guiding principle was "all buildings must satisfy three criteria: strength (*firmitas*), functionality (*utilitas*), and beauty (*venustas*)" (Rowland, Howe, and Others, 2001). In a way, Vitruvius presented the concept of functionalism more than twenty centuries before modernism (Bürdek, 2005).

In the 18th century, during the industrial revolution, the designer role took a prominent place. Mass production required a more defined division between design and manufacturing (Sparke, 2009). This divide was the beginning of *industrial design*: objects were designed by someone and built by someone else. The inception of assembly



Figure 2.2: Parthenon by jhadow. Licensed under .

lines required workers focused on individual tasks to replicate a given design over and over (Loewy, 2002). For example, the iron pot introduced by Abraham Darby in 1907 was one of the first mass-produced industrial objects (Figure 2.3). He presented a new way of casting iron using a master pattern and sticky green sand. The same pattern was used repeatedly to create multiple molds.



Figure 2.3: The revolutionary cast iron cooking pot. Copyright Ironbridge Gorge Museum Trust.

On the other hand, the Arts & Crafts movement saw mass-production as responsible for the deplorable living conditions of some workers (Stansky, 1985). William Morris, a supporter of this movement, tried to put back the human into the design and manufacturing process. Morris opposed mass production and built products based on

craftsmanship and handmade processes. However, this distinction between handmade and mass-produced was not always clear. For example, designers such as Joseph Hoffmann wanted to move away from the ornaments of the past, so they created objects that looked modern and geometric. There were sometimes hand built and sometimes created with the help of machines at low scale batches.

In the early 1900s, people such as Van de Velde from the Art Nouveau movement rooted for the individuality of the artist, while others such as Peter Behrens from AEG focused on designing for the industry and seeking standardization. A typical example of this "industrial design" is the development of the car industry for a mainstream audience. Henry Ford wanted a vehicle that was lighter, simpler to use and inexpensive to manufacture. He did not invent the car, but he worked with a group of designers to create the popular model T (Figure 2.4). Ford was obsessed with improving the manufacturing process of the car. For example, constructing a single model T went from 12.5 working hours to 93 minutes after the introduction of three parallel assembly lines (Evans, Buckland, and Lefer, 2009). J Mays, former Global Design Chief at Ford, said in an interview in 2009 that the development of the car industry created a division between car designers, in charge of the body, and car engineers, in charge of the chassis (Sparke, 2009).



Figure 2.4: 1914 Ford Model T. Four Cylinder, Twenty Horsepower by LibertyGroup25. Licensed under 

Several design movements and trends can be seen throughout history. For example, the search for standardization and functionalism gave birth to the Bauhaus School of Design. Technological advances such as the electrification of the home, the telephone, and the use of plastic materials gave designers a vast playground for new ideas. For a more in-depth review see Walker and Attfield (1989).

2.2 SOFTWARE PROCESS MODELS

We can draw a parallel between the history of design and prototyping on the one hand, and the design and prototyping in software engineering on the other hand. Initially, "software artisans" encompassed the design and construction of the system, where the user interface was created as an afterthought (Cooper, 2004). Later on, in the same way that car designers focused on the car's body and car engineers focused on the car's chassis, a similar dichotomy occurred between user interface design and software development. This led to the introduction of multiple methodologies to organize the software development process.

2.2.1 Waterfall model

The first software development methodology, the Waterfall Model, was introduced at a symposium on advanced programming methods for digital computers in June 1956 (United States. Navy Mathematical Computing Advisory Panel, 1956). Benington presented how the MIT Lincoln Laboratory organized the production of the programs for the SAGE (Semi-Automatic Ground Environment) system (Figure 2.5).



Figure 2.5: The SAGE system. Information is sent to the *direction center* (in the middle) from multiple sources. At the top, from left to right: long-range radars, airborne early-warning planes, radar picket ships, gap-filler radars, offshore air defense radars, air traffic control information. At the bottom, instructions can be sent to missiles and aircraft after processing the information. Image from the Computer History Museum (*In Your Defense*).

SAGE was an air-defense system used by the United States Air Force to defend against potential mass jet bomber attacks (*In Your Defense*). The system collected data from several radars and calculated

information about potential enemy aircraft, such as speed, altitude, and direction.

If the aircraft was identified as hostile by human operators, the system was able to guide manned interceptors or missiles to the target (*In Your Defense*). The construction of such large computer programs integrated with multiple subsystems required a structured top-down approach, flowing in one direction like a waterfall. In a traditional waterfall process, compartmentalized phases are executed sequentially, forcing one phase to finish in order to start the next one. For example, the coding only begins after a complete specification is available (Figure 2.6).

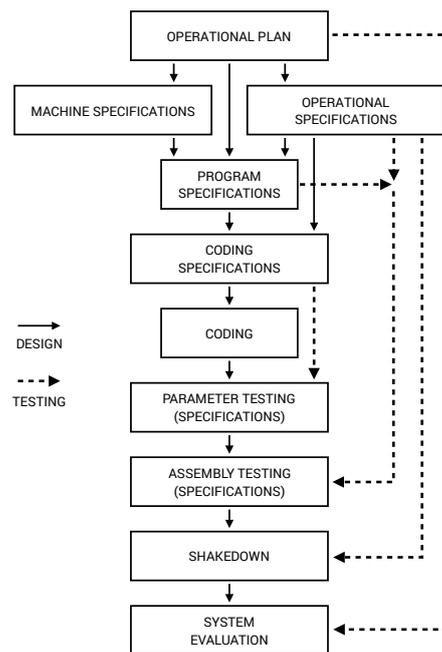


Figure 2.6: The waterfall process used during the construction of the SAGE system. Image reproduced from Benington (1983).

Benington (1983) republished the original presentation of 1956 but added some notes with two new interesting pieces of information. First, he states: "it is easy for me to single out the one factor that I think led to our relative success: we were all engineers and had been trained to organize our efforts along engineering lines". This shows how important it is to share the same value system among the stakeholders involved in this rigid process. Second, Benington (1983) clarifies (emphasize mine): "I do not mention it in the attached paper, but we undertook the programming only after we had assembled an *experimental prototype* of 35,000 instructions of code that performed all of the bare-bone functions of air defense". This illustrates the importance of prototyping even for proponents of a waterfall-like process, where coding should not start until a plan is already in place. Benington (1983) goes even further (emphasize mine): "*the biggest*

mistake we made in producing the SAGE computer program was that we attempted to make too large a jump from the 35,000 instructions we had operating on the much simpler Whirlwind I computer to the more than 100,000 instructions on the much more powerful IBM SAGE computer" (Benington, 1983). This denotes the need not only for an incremental process, i.e. where the system is built in pieces but also for an iterative process, i.e. where the system is evolved through successive refinements (Figure 2.7).

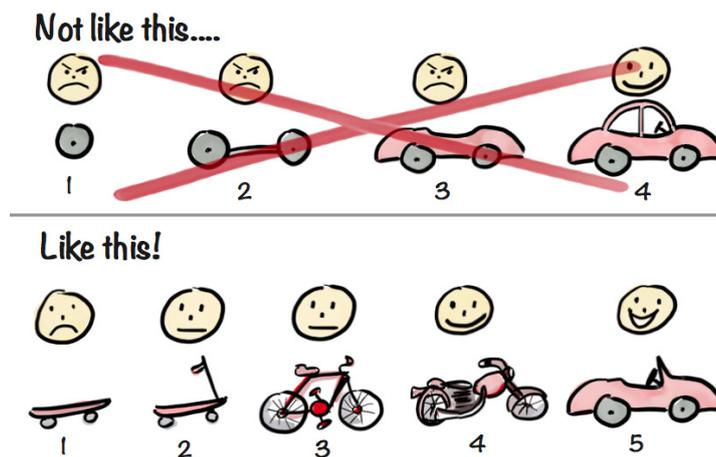


Figure 2.7: Incremental vs Iterative. Copyright: Henrik Kniberg <https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp>

A waterfall model can be successful in the production of software products when plans are rigid. Architects have been using variants of this process for centuries, where the plan needs to be finished before construction starts. In 1968 and 1969, during the NATO Software Engineering Conferences, terms such as "software crisis" and "software engineering" were introduced, with the waterfall model as their flagship process (Buxton and Randell, 1970). The success of this model relies on the predictability of the requirements and the irrevocability of the plan. However, when changes are introduced in a later phase, they have an impact in all the preceding phases. In other words, the more advanced the process, the higher the cost of introducing a change.

The software medium is more malleable than traditional materials used by classic designers (Kay, 1984). Software engineers soon realized that having irrevocable plans was not an easy constraint to enforce. In the book *The Mythical Man-Month: Essays on Software Engineering*, Brooks (1975) says that "the first system built is barely usable [...] hence plan to throw one away; you will, anyhow". One of the most popular works about waterfall-like methodologies is Royce (1987). However, Royce proposed a modified version of the waterfall model where phases are not only flowing in one direction but "there is an iteration with the preceding and succeeding steps". In the same line as Brooks (1975), Royce recommends going through the process twice,

first as a miniature version of the real complete process as a kind of simulation.

2.2.2 *Agile Methodologies*

In the late 1980s, there was a generalized concern that the waterfall model was discouraging well-established engineering practices such as prototyping and reuse (Brooks and Others, 1987). Boehm (1986) introduced the *Spiral model* to overcome these issues by combining the strengths of other approaches, such as *the evolutionary development model* (McCracken and Jackson, 1982) and *the transform model* (Balzer, TE Jr, and Green, 1983). In the spiral model, each cycle of the spiral has the same sequence of steps, with variants for each part of the product and each level of elaboration.

The first software crisis (Buxton and Randell, 1970) was characterized by the inability to construct large and complex programs, e.g., with assembly programming languages, that met budget and time expectations. This crisis was approached from a technical perspective, with the introduction of better tools such as high-level languages, e.g., Fortran and C, and from a methodological perspective, with the widespread adoption of waterfall-like models. The second software crisis, in the 1980s and 1990s, was characterized by the inability to maintain and extend these large and complex programs (Glass, 2006). This new crisis was also addressed with better tools, such as object-oriented programming (Dahl, 1970; Goldberg and Robson, 1983), and better software engineer practices, such as design patterns (Gamma et al., 1994) and agile methodologies (Beck et al., 2001).

In the 1990s, there was a shift from seeing software engineering as a purely rigid construction process to a more flexible service-oriented view. Proponents of agile methodologies, such as Beck (1999) with Extreme Programming or XP (Figure 2.8), wanted software to meet user expectations while allowing changes to be made more easily without incurring unsustainable costs. There are several *flavors* of agile methodologies, such as XP or Scrum, but it is out of the scope of this work to review them all. However, the main idea shared among the various agile methods is the use of an incremental and iterative approach, generally seeking "small" iterations (Figure 2.7).

2.2.3 *Participatory Design*

The Xerox Alto was the first personal computer with a tied integration between an operating system and a Graphical User Interface (GUI) (Thacker, MacCreight, and Lampson, 1979). Until then, software was mainly designed by developers for developers. The new personal computer created at Xerox PARC (Palo Alto Research Center) focused on another type of users, i.e. secretaries. Designers at

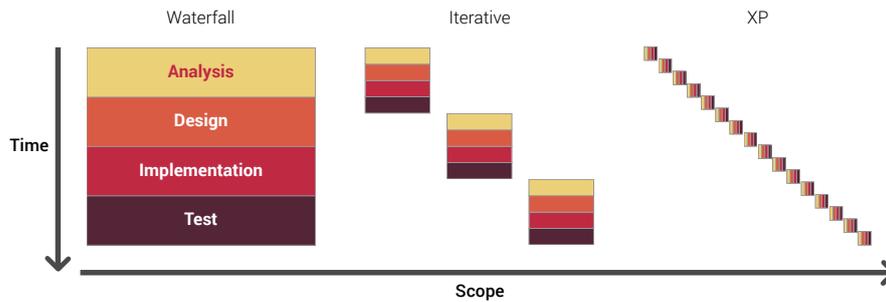


Figure 2.8: The Waterfall Model has long development cycles (analysis, design, implementation, test) while the iterative methods, such as the Spiral Model, mix these activities in a shorter development cycle. XP blends all the activities, a little at a time, during the entire software development process. Figure based on the model presented in Beck (1999)

PARC investigated how secretaries worked and how they would use a futuristic GUI before starting any engineering effort (Smith et al., 1982). Files, inboxes, and documents were the main objects that the user would manipulate. In a way, the Alto computer was one of the largest software prototypes ever built. The Alto was used as a research prototype for more than seven years before releasing a commercial version of the personal computer called the Xerox Star (Smith et al., 1982).

However, the approach taken by these designers was not the most common approach in the industry. Regardless of the chosen methodology, few software projects considered the design and construction of user interface as an essential part of the development process. Many examples of bad interfaces have been documented (Johnson, 2000), where user interface design is as an afterthought or is only delegated to software developers. For example, Cooper (2004) mentions that in early Windows systems, one of the reasons for using more dialog boxes than drag & drop interactions was just because the former required fewer lines of codes to implement than the latter. When user interface decisions are only in the hands of developers, the developer's convenience will compete with the user needs.

The Co-operative Design Methodology involves actual users early in the design process. Between 1981 and 1985, researchers from Denmark, Norway and Sweden worked on a project called UTOPIA (acronym for "Training, Technology And Product In Quality of work perspective" in Danish, Norwegian and Swedish) (Bødker et al., 2000). This project represented an early development of Co-operative Design, the goal of which was to "give the end user a voice" (Bødker et al., 2000). They created a "technology laboratory" where different designs could be materialized with low tech tools such as wooden mice, paper menus and "a graphic workstation for illustrating prototypes of computer based tools" (Bødker et al., 2000). In the 1990s these methods became a success in the United States (Bødker et al., 2000) as Participatory De-

sign (Greenbaum, 2017; Muller and Kuhn, 1993). Prototyping was no more dedicated only to expert engineers. Thanks to low-tech methods a broader audience was able to participate in the design process.

Around the same time, interaction design emerged as a response to the poor interfaces commonly encountered in computer systems (Johnson, 2000). Interaction design is informed by product design and software engineering (Hartmann, 2009). Moggridge and Atkinson (2007) explain that "in the same way that industrial designers have shaped our everyday life through objects that they design for our offices and for our homes, interaction design is shaping our life with interactive technologies—computers, telecommunications, mobile phones, and so on". Combining agile methodologies and interaction design is not an easy task (Silva et al., 2011). Both share similar artifacts and goals, e.g., how to go from an ill-defined problem to a product that satisfies the user needs. However, software agile methodologies bring the user closer to the engineering process while interaction design, and in particular Participatory Design, brings the user closer to the design process (Beck and Cooper, 2002). On the other hand, both approaches agree on having an incremental and iterative process. Regardless of the selected methodology, carefully selecting the right prototyping approach and the desired characteristics of the created prototype are essential.

2.3 PROTOTYPING AND PROTOTYPES

There are multiple processes to guide the production of digital and interactive products. In this dissertation, I am focusing on the prototyping of interactive systems.

2.3.1 Prototyping as a process

Floyd (1984) sees prototyping as a key component of any stage of the software development methodology. This collides with the approach proposed by the phase-oriented approaches, such as the Waterfall model, that have a strict order for the prototyping activity. Floyd (1984) emphasizes that prototyping for interactive systems "enhance the communication between developers and users concerning the suitability of man-machine interfaces".

As an extreme example of how prototyping can improve the communication between stakeholders, Hertzfeld (1982) tells the story of "the Steve Jobs Roll Your Own Calculator Construction Set". In the early 1980s, Chris Espinosa was working on the Calculator's visual design for the Macintosh:

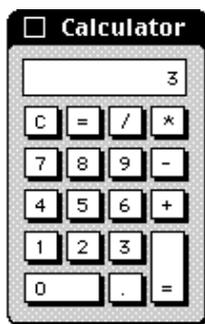


Figure 2.9:
Calculator from
Hertzfeld (1982)

After playing around for a while, he [Chris] came up with a calculator that he thought looked pretty good [...] We all gathered

around as Chris showed the calculator to Steve and then held his breath, waiting for Steve's reaction. "Well, it's a start", Steve said, "but basically, it stinks. The background color is too dark, some lines are the wrong thickness, and the buttons are too big.

So, for a couple of days, Chris would incorporate Steve's suggestions from the previous day, but Steve would continue to find new faults each time he was shown it. Finally, Chris got a flash of inspiration.

The next afternoon, instead of a new iteration of the calculator, Chris unveiled his new approach, which he called "the Steve Jobs Roll Your Own Calculator Construction Set". Every decision regarding graphical attributes of the calculator were parameterized by pull-down menus. You could select line thicknesses, button sizes, background patterns, etc. (Hertzfeld, 1982)

Jobs started playing around with this interactive prototype to find the combination of visuals that satisfied him. The story goes that this design was the one that shipped with the Macintosh and it remained the default Calculator for many years (Hertzfeld, 1982).

2.3.1.1 Prototyping steps

Floyd (1984) characterizes four steps in prototyping:

1. Functional Selection, i.e. choosing the functions to prototype;
2. Construction, i.e. implementing the prototype;
3. Evaluation, i.e. analyzing the prototype with relevant user groups;
4. Further Use, i.e. discarding the prototype or using it as part of the final system.

Functional Selection requires deciding the functional *scope* of the prototype, i.e. vertical or horizontal prototyping (Figure 2.10). In vertical prototyping, only a few selected functions are implemented as in the final system. While in "horizontal prototyping", the selected functions are not implemented in detail but many functions are demonstrated. There are many tools for the *Construction* of prototypes that I will describe in Section 2.3.3, but the goal should be to reach the *Evaluation* step quickly rather than implementing for long-term use.

Floyd (1984) also highlights that the prototyping process should be designed as a learning process, in order to use the prototypes as "learning vehicles". For this reason, a prototype should be available early in the development process "to offer full benefit to all parties concerned: developers, customers, and users (hence the term 'rapid prototyping')" (Floyd, 1984). Another aspect is that the prototype should work in the context of the user's task, i.e. "it should involve

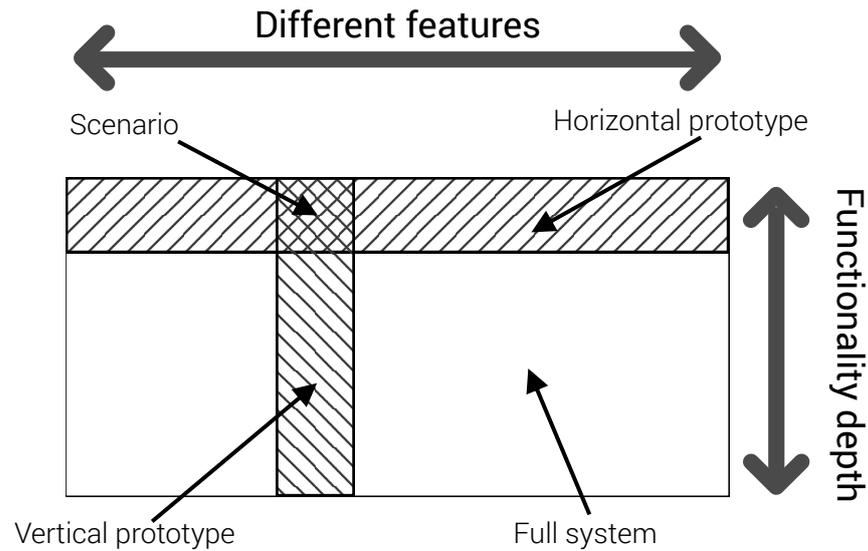


Figure 2.10: Horizontal and Vertical prototyping. Horizontal prototyping covers multiple features but disregards depth of the functionality. On the other hand, vertical prototyping provides full depth of functionality of a minimal number of features. Figure reproduced from Nielsen (1994)

authentic and nontrivial problems" (Floyd, 1984). Finally, a prototype needs to be easy to modify, i.e. "there should be an easy way of changing the prototype by revising existing or adding new features as needed" (Floyd, 1984).

2.3.1.2 Prototyping approaches

Prototyping can have different goals, Floyd (1984) distinguishes between the following classes of prototyping¹:

1. Exploratory Prototyping;
2. Experimental Prototyping; and
3. Evolutionary Prototyping.

Exploratory prototyping is focused on the early stages of the development process. This class of prototyping increases the knowledge of the developers about the domain of the problem and distills the users' ideas of what a computer system might do for them. This type of prototyping creates practical demonstrations of future systems that are very informal in order to explore alternatives and promote a creative co-operation between all the stakeholders. Floyd (1984) only recommends this class of prototyping "if there are tools available which keep to a minimum the effort required in constructing the prototype".

¹ This is known as the "the triple E model"

Experimental prototyping refers to the idea of exposing the prototypes to the actual users. According to Floyd (1984), the term "experimental" should not be taken in a technical sense, because the conducted experiments are "soft" and not based on rigorous theories. The main goal is to gather feedback from the user.

Finally, evolutionary prototyping acknowledges that contexts and requirements will change. The introduction of a new interactive system in an environment will most likely modify the context of use and thus require changes in the interactive system itself. To support the evolution of the prototype to these new changes, Floyd (1984) outlines an incremental and iterative process similar to the agile methodologies approach.

2.3.1.3 Prototyping in other disciplines

In the 1980s, prototyping was a recent development in information system design. Janson and Smith (1985) compared prototyping in mechanical and electrical engineering with prototyping in systems engineering (Figure 2.11).

They found that the major differences are "the lack of tightly written systems design specifications and the short time period required to provide the user with an initial system for actual 'hands-on' experience". They divided the development life-cycle of both, mechanical engineering and system design into three stages: *intelligence*, *design* and *implementation*. They also proposed the following categories of prototypes:

- Real-life, i.e. full-scale representation using finalized materials;
- Simulated, i.e. using a different medium of construction than the final materials; and
- Real-life/Simulated, i.e. a mix of real-life and simulated parts.

Table 2.1 shows the relationship between the category of prototypes and its application to the corresponding stage in the development life-cycle. As expected, prototypes with low to medium relative cost are used in early stages while more costly prototypes are more adequate at the end of the design phase and the start of the implementation phase. According to Janson and Smith (1985), simulation prototypes are the most applicable when prototyping for design. However, simulation prototypes require materials different from the final system. Existing techniques from other fields might provide these alternative mediums for creating simulation prototypes.

Wong (1992) encourages interface designers to borrow techniques from graphic design. Wong (1992) argues that coded prototypes are too time-consuming and do not facilitate early-stage design decisions. The finalized look of a coded prototype can mislead designers into thinking that a drafted concept is a nearly finished product. This

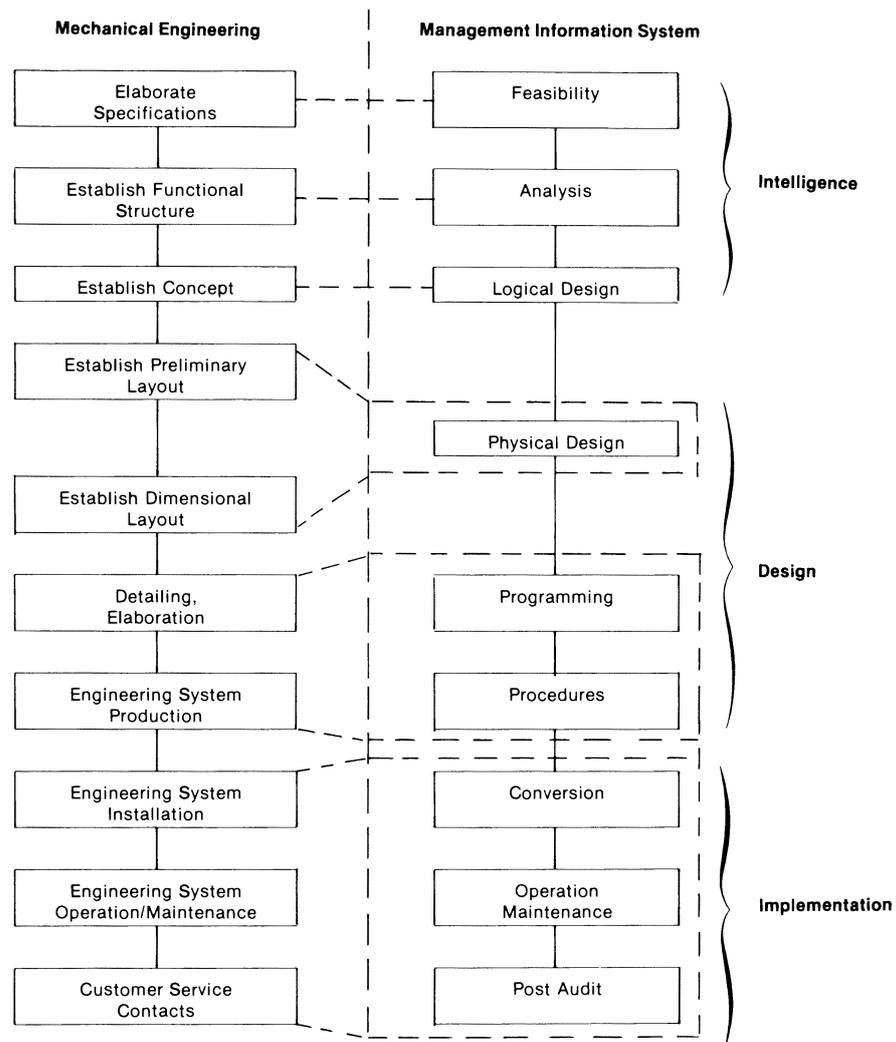


Figure 2.11: The system development life-cycle in mechanical engineering compared with information system engineering. Figure from Janson and Smith (1985)

echoes some of the recommendations of Thompson and Wishbow (1992), which for example propose to use regular computer-based painting programs as prototyping tools. Stolterman et al. (2009) also encourage interaction designers to create "designerly tools" in order to support prototyping with instruments closer to the design activity.

2.3.2 Prototypes as artifacts

While Floyd (1984) presented a process view of prototyping, i.e. steps and approaches for prototyping, Bäumer et al. (1996) presented a product view, i.e. a classification of the kinds of prototypes that can be created:

- Presentation Prototypes focus on the UI to illustrate how the system solves a problem;

Prototype Use	Where Used in Life Cycle Stage	Real- Life	Simu- lation	Real- Life / Simu- lation	Relative Cost
Requirement Specification	Intelligence (at the end) or Design (at the start)	+	++	+++	Low to Medium
Design Selection Understanding Test System Components	Intelligence or Design	+	+++	++	Low
Test Evaluation Completed System	Design (at the end) or Implementation (at the start)	+++			High

Blank; not applicable
+ Least Applicable
++ Applicable
+++ Most Applicable

Table 2.1: The Application of Prototyping in the Design Process. Table from Janson and Smith (1985)

- Functional Prototypes implement part of the UI as well as the functionality of the application;
- Breadboards focus on technical aspects, i.e. they are built to evaluate the technical risk, not the user interactions;
- Pilot systems are almost finished products that can be deployed in real contexts of use.

Bäumer et al. (1996)'s product view of prototypes can be linked with Floyd (1984)'s triple E model. Designers are expected to generate Presentation or Functional Prototypes for Exploratory Prototyping, Functional Prototypes and Breadboards for Experimental Prototyping, and Pilot Systems for Evolutionary Prototyping.

Houde and Hill (1997) argue that we lack a language to talk about prototypes. They define a prototype "as any representation of a design idea, regardless of medium" and a designer "as anyone who creates a prototype in order to design, regardless of job title". They argue that the prototypes need to be more focused and propose a classification based on their purpose rather than on the prototype itself. They propose a three-dimensional space that relates to typical questions

that designers want to answer when creating prototypes: role, look and feel, and implementation (Figure 2.12). "Role" refers to questions about the usefulness of the product in the user's life, and it usually requires the context of use to be well established. "Look and feel" refers to questions about the experience of using the product, i.e. "the concrete sensory experience" (Houde and Hill, 1997), and it usually requires a simulation of the experience. "Implementation" refers to questions about how the product work internally and usually requires a working system. Different prototypes can be positioned in this triangular space, closer or farther away from the vertices.

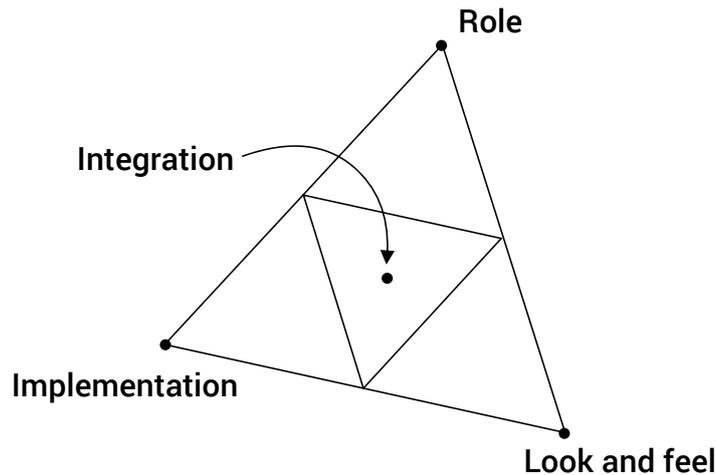


Figure 2.12: What do prototypes prototype. Figure from Houde and Hill (1997)

Similarly, Buchenau and Suri (2000) introduce "Experience Prototyping". While there are similarities with the dimensions presented by Houde and Hill (1997) and others, "Experience Prototyping" goes beyond the concrete sensory experience and tries to transform a passive audience into active participants. Buchenau and Suri (2000) define "Experience Prototyping" as "any kind of representation, in any medium, that is designed to understand, explore or communicate what it might be like to engage with the product, space or system we are designing".

2.3.3 *Low vs High Fidelity Prototypes*

One classification that is as popular as controversial is the dichotomy between low- and high-fidelity prototyping. Fidelity refers to how accurate the prototype represents the final system for the user. This contradicts the idea of Exploratory Prototyping: designers should not be able to assess the fidelity of a system that is still being "discovered". However, in less open-ended scenarios, the universe of potential designs directions is known beforehand.

Another critique of the low-high classification is the lack of a clear dimension of analysis. Cooper (1995) describes three main overlapping concerns in user experience design: *form* –industrial and graphic design–, *behavior* –interaction design–, and *content* –information architecture and copywriting–. For example, a paper prototype (Rettig, 1994) is generally considered a low-fidelity prototype, but it can represent the *content* of the system with a high level of fidelity and, at the same time, have a low level of fidelity for the *form*.

There is an implicit association between low-fidelity prototyping and rapid prototyping. Gordon and Bieman (1995) call rapid prototyping the creation of prototypes early in the software development life cycle. Also, they found that rapid prototyping can be effectively used by software developers and can improve the alignment between the functions of the product and the needs of the users. It is assumed that the closer the prototype is to the envisioned final product, the more costly it is to create. For this reason, there is an implicit relationship between low-fidelity prototypes and rapid prototyping techniques. The outcome of a rapid prototyping technique is generally considered a throwaway prototype:

"That's right. Tear it into little pieces and start again. Don't expect it to be right the first time. You don't know your audience well enough. You do not fully know the limits of the prototyping tool. You will not know the application inside and out, and you don't know whether the developers can implement what you have prototyped. Prototyping is an iterative process, and you are going to learn as you go along. Leave enough time in your schedule to make radical changes based on the feedback you receive."

Rudd and Isensee (1994).

Campos and Nunes (2007) surveyed 370 practitioners and found that low-fidelity were the most used prototypes in every development process and that there is a trend towards using more informal tools.

Virzi, Sokolov, and Karis (1996) conducted a study to compare the usability problems found with a low- and high-fidelity prototype of an electronic book and an Interactive Voice Response (IVR) system. They found that participants detected the same amount of usability studies with both types of prototypes. More recently, Lim et al. (2006) analyzed three prototypes for a mobile messaging application: paper, computer-based and fully functional prototype. Participants found the most common usability issues in the three prototypes but some issues, such as physical handling and performance-related problems, were only identified on the computer-based and fully functional prototypes.

Both types of prototyping have advantages and disadvantages (Rudd, Stern, and Isensee, 1996). For example, low-fidelity prototypes are cheap to create and provide a meaningful range of interaction styles

with simple techniques (Hall, 2001). However, this same expressiveness can generate a "wishful thinking" mindset, making designers prototype ideas that are impossible to implement or that do not solve an actual user problem (Holmquist, 2005). Practitioners know that low and high fidelity should not be considered the only options available, and mixing different levels of fidelity in the same prototype is a common practice (McCurdy et al., 2006).

In summary, while practical, the low/high classification of prototypes is also simplistic and should be used carefully. Designers need to consider many more dimensions for analyzing the characteristics of their prototypes.

2.3.4 *Prototyping Dimensions*

Beaudouin-Lafon and Mackay (2003) propose four dimensions to analyze prototypes and prototyping tools:

- Representation, i.e. the form of the prototype;
- Precision, i.e. the level of detail to evaluate the prototype;
- Interactivity, i.e. to which extent the user can interact with the prototype; and
- Evolution, i.e. the life-cycle of the prototype, e.g. iterative or throwaway.

They divide representation between paper prototypes (off-line) and software prototypes (on-line). Paper prototypes are inexpensive and accessible to a wide range of participants, making them more suitable for the early stages of the design. However, paper prototypes limit the exploration of certain types of interfaces that are dynamic. To add interactivity to these off-line prototypes, designers use the **WOz** technique to simulate the system responses.

Software prototypes provide more interactivity but they often require coding or they constraint the design alternatives, e.g., by using a **UI Builder** only certain widgets are available. Unfortunately, early-stage prototyping tools should not limit the designers' options, i.e. the goal is to explore alternatives.

Precision and fidelity are two related terms; the first means "relevance of details with respect to the purpose of the prototype" (Beaudouin-Lafon and Mackay, 2003) while the second focuses on the "closeness to the eventual design" (Houde and Hill, 1997). The purpose of the prototype defines the correct level of precision: relevant details should have a high precision while irrelevant details could be left open with lower precision.

Interactivity refers to the levels of the interaction supported by the prototype. Beaudouin-Lafon and Mackay (2003) define three levels of interactivity: fixed, fixed-path and open. Fixed prototypes are

non-interactive, such as a video clip. Fixed-path provides minimal interactivity, such as using presentation software to simulate with sequential slides the transition among pre-defined screens. Open prototypes focus on a particular area of the prototype and provide a level of interaction limited but similar to the final system. For example, a computer-based prototype created with a scripting language can provide multiple interactive paths.

Finally, evolution describes the life-cycle of the prototype. Rapid prototyping generally produces throwaway prototypes that have a short life-cycle and are discarded once they have served their purpose. Iterative prototypes are reused to build on top of previous iterations. A particular case of iterative prototypes is an evolutionary prototype, which is iterated until it becomes the final system.

Lim, Stolterman, and Tenenberg (2008) present an *anatomy of prototypes* that analyzes prototypes along two main dimensions:

"Prototypes are filters that traverse a design space and are manifestations of design ideas that concretize and externalize conceptual ideas." (Lim, Stolterman, and Tenenberg, 2008).

I agree with this view of prototypes as *filters* and *manifestations*. Lim, Stolterman, and Tenenberg (2008) mention some filtering dimensions such as appearance, data, functionally, interactivity and spatial structure. In this dissertation, I am grouping concerns such as appearance and spatial structure under the notion of *form*. Lim, Stolterman, and Tenenberg (2008) also mention some manifestations dimensions such as material, resolution and scope. Scope relates to the previously mentioned vertical vs. horizontal prototyping (Figure 2.10). Finally, we can draw a parallel between Beaudouin-Lafon and Mackay (2003) dimensions and Lim, Stolterman, and Tenenberg (2008) manifestations, such as the similarity between representation and material, and precision and resolution.

In this dissertation, the main prototype representations that I am handling are paper and computer-based. To narrow down the precision of a prototype we need to specify the dimension of interest, i.e. content, form or behavior (Cooper, 1995). In terms of interactivity, we can have non-interactive prototypes, fixed-path and open prototypes. While paper prototypes and video clips are traditionally considered non-interactive, video provides a higher level of interactivity than paper when it is not combined with the WOz technique. For this reason, I am adding a fourth category named "only watchable", i.e. the prototype does not possess the interactivity of the final system but can be paused/resumed/rewound to mimic certain interactions. In terms of evolution, I am focusing on rapid/throwaway and iterative prototypes. I synthesize these dimensions with 14 examples of prototyping tools and techniques in Figure 2.13. Standalone paper prototyping

	Evolution	Representation	Precision (Low - Medium - High)			Interactivity
			Content	Form	Behavior	
PEN AND PAPER						none
PHYSICAL MOCK-UPS		Paper Prototyping	L	L ⁺	L ₊	none
WIZARD-OF-OZ						fixed-path / open
VIDEO PROTOTYPING						none / only watchable
NON-INTERACTIVE SIMULATION						none / only watchable
INTERACTIVE SIMULATIONS		Software Prototyping	L ₊	MH	M ₊	fixed-path / open
SCRIPTING LANGUAGES						
GRAPHICAL LIBRARIES						-
WINDOW SYSTEMS		Software Tools	M	MH	LH	open
UI TOOLKITS						
UI BUILDERS						
APPLICATION FRAMEWORKS						
MODEL-BASED TOOLS		Software Environments	MH	MH	LH	open
UIDES						

Figure 2.13: Table based on the dimensions presented by Beaudouin-Lafon and Mackay (2003) for analyzing prototypes: evolution, representation, precision, and interactivity. Precision is subdivided into the three interaction design dimensions presented by Cooper (1995): content, form and behavior. A range of precision covers one or more of the following levels: Low, Medium and High. For example, LH denotes a range from low to high, i.e. including medium, while L is only low. Plus (+) and minus (-) signs highlight small variations on the precision for that particular technique, i.e. higher or lower respectively.

does not support interactivity. Pen-and-paper can support limited interaction when it is combined with the *WOz* technique. To support more dynamic interactions, designers are forced to either use software prototyping, usually restricted to certain interaction styles, or program with code, limiting the participation of non-developers.

In summary, previous research analyzed prototyping from a process view and a product view. From a process view, prototyping serves different purposes, ranging from the exploration of ideas to the evaluation of functional systems. From a product view, prototypes *filter* and *manifest* different characteristics of the system under design.

This dissertation investigates early-stage software prototyping tools that let designers:

- reuse previous prototypes to iteratively explore multiple variations; and
- prototype high levels of interactivity as fast as non-interactive paper prototyping.

Part I

VIDEO PROTOTYPING

WHY VIDEO PROTOTYPING?

Pen-and-paper is widely used when designing and communicating interactive systems, especially for rapid prototyping (Beaudouin-Lafon and Mackay, 2003). Sketching on paper has well-known benefits (Buxton, 2007): it does not require technical skills, is inexpensive –in time and money– and, as a consequence, is easy to throw away. Paper excels in representing static visual properties and physical transformations such as moving paper elements (Greenberg et al., 2012). However, paper has some limitations; for example, it is difficult or impossible to create dynamic transformations that continuously re-shape or modify the design elements, such as re-sizing or stretching elements or modifying colors and feedback in response to continuous user input.

During a paper prototyping session (Snyder, 2004), a user interacts with the prototype, while one or more designers, or *wizards*, play the role of the computer. With this technique, the prototype is depicted by a collection of paper elements ranging from imprecise hand-drawn sketches to highly refined printed images. When the design changes or when exploring variants, instead of modifying the existing paper representations, they are thrown away and new ones are quickly created. The user can simulate the interaction over the paper prototype to communicate a rough idea, such as tapping with a finger to simulate a mouse click instead of using an actual indirect input device. The Wizard of Oz (WOz) technique (Green and Wei-Haas, 1985) can create more realistic prototypes when the wizards conceal their actions. The WOz technique is not limited to paper and can be used, e.g., with a video projector to create a more compelling setup. While both paper prototyping and WOz can also be used for user evaluation with real users, this dissertation only focuses on their application during early-stage prototyping.

Video prototyping (Mackay, 1988; Mackay, 2002; Mackay and Falyard, 1999) combines paper and video with the WOz technique to capture interaction, and to communicate and reflect about interaction design. Videos can range from an inexpensive recording of a traditional paper prototyping session (Rettig, 1994) to a high-budget video prototype (Tognazzini, 1994) requiring specialized equipment (Bardram et al., 2002). In this work, I focus on using video during early-stage design (Vertelney, 1989) rather than during other research activities (Mackay et al., 1988). Video is an extremely flexible medium that accommodates several aspects of the design, such as the user interface, the user interactions, and even the context of use. Video provides additional prototyping capabilities, such as jump cuts for sim-

ple appear/disappear effects or adding shots for contextualizing the user and the system within a story. However, despite these qualities, traditional video prototyping has some limitations.

Firstly, while it is reasonably easy to make corrections or discard paper sketches, video is not as forgiving. Shooting video is a time-consuming activity: careful planning is required to coordinate multiple designers working simultaneously as writers, directors, scenographers and actors. Also, current video digital tools are not specifically targeted to interaction design, leaving the planning as an ad-hoc process, or only focusing on post-hoc video editing. In other words, current tools encourage designers to capture as much as possible and craft the final artifact in post-production. However, video prototyping belongs to the rapid prototyping family of techniques and should not rely on a laborious post-production process to produce a quick prototype. How can digital video tools for interaction design help designers rapidly create video design artifacts within a single design session?

Secondly, using video together with paper hinders some of the benefits of using paper alone. Depending on the audience of the video, the wizard's trickery might need to be concealed, increasing the time and cost to produce a prototype. Introducing changes in the paper prototype creates inconsistencies with previously recorded scenes, leaving designers with three choices: sacrificing the consistency throughout the video, fixing the affected scenes in post-production editing, or re-shooting all the affected scenes. How can we preserve the consistency of the design, avoid costly post-production edits and reduce repetitive re-shooting?

My goal is to provide better support for video prototyping by creating digital tools that aid during the planning and composition of the video prototype artifact. First, I present related work on early-stage prototyping tools and techniques with a focus on video-based tools. Then, I report on a questionnaire study that investigate the current barriers that designers encounter when using video for prototyping.

3.1 RELATED WORK: EARLY-STAGE PROTOTYPING TOOLS AND TECHNIQUES

In recent years many academic and commercial tools have emerged to support the prototyping of graphical user interfaces (Silva et al., 2017). While pen-and-paper is one of "the most widely used prototyping medium" (Carter and Hundhausen, 2010), some researchers argue that informal computer-based tools might better support the prototyping of interactive behaviors (Bailey and Konstan, 2003). In the following literature review, I am reviewing early-stage commercial tools¹ and then focus on video-based tools.

¹ I will cover academic computer-based tools in [Part ii](#).

3.1.1 Commercial tools

Commercial tools have evolved from the graphic design tradition, starting from sketching tools but currently focusing more on "pixel-perfect" designs (Vinh, 2015), with graphic-authoring tools such as Adobe Illustrator or Bohemian Sketch (Bohemian BV, 2009). However, most of these tools do not target early-stage design as they focus on the final *look* rather than the *feel*. The few commercial tools that support custom behaviors require visual or textual programming skills, such as Origami (Facebook, 2013) or Framer (Framer BV, 2015). Lee et al. (2017) observe that much of the interactive behavior remains as textual descriptions due to the cost of creating dynamic prototypes, even for professionals.

Some tools extend traditional graphics authoring to support animations and effects, such as Flinto (2010), but they ignore the role of user inputs and contexts of use. Only a handful of commercial tools support informal early-stage prototyping, e.g., by using paper-in-screen techniques (Bolchini, Pulido, and Faiola, 2009). One of these tools is POP, which lets designers create simple screen-flows by connecting pictures of paper prototypes through digitally defined hotspots or "actions points" (Marvel Prototyping Ltd, 2012). However, while POP supports multiple transition animations, these can only be executed through discrete actions, e.g., a finger tap.

To mimic dynamic interactions many designers use presentation software such as Microsoft PowerPoint or Apple Keynote (Khella Productions Inc, 2013). While this is suitable for some use cases, e.g., Windows, Icons, Menus, Pointer (WIMP) and mobile applications, the predefined animations only include effects and transitions among slides, thus covering a tiny subset of all the available dynamic effects that designers might want to explore.

Professional designers also use video editing software, such as Adobe After Effects, to prototype the *look* of continuous interactions with high visual fidelity videos. For example, Luciani and Vistisen (2017) use animation-based sketching techniques with professional editing tools, such as Adobe Premiere. However, current approaches to video editing are complex and time-consuming, which conflicts with the goals of early-stage prototyping. For example, VideoSketches (Zimmerman, 2005) uses photos instead of videos just to avoid the high cost and production issues of creating video scenarios.

The benefits of video as a design tool have been investigated for a long time (Mackay et al., 1988). More recently, Wong and Mulligan (2016) have studied the use of concept videos not only to illustrate future artifacts but also to embed the design in a broader context. Interestingly, Dhillon et al. (2011) have found no differences in the quality of feedback between a low-budget fidelity and a high-budget video prototype.

In summary, current commercial tools support the creation of refined prototypes, and are therefore more appropriate for mid/late stages of the design, while early-stage tools lack features to explore the details of continuous interaction. In this dissertation, I explore how to provide an enhanced video prototyping workflow as quick as low-budget video prototyping but also featuring capabilities to illustrate custom dynamic interfaces found in high-budget video prototyping (Figure 3.1).

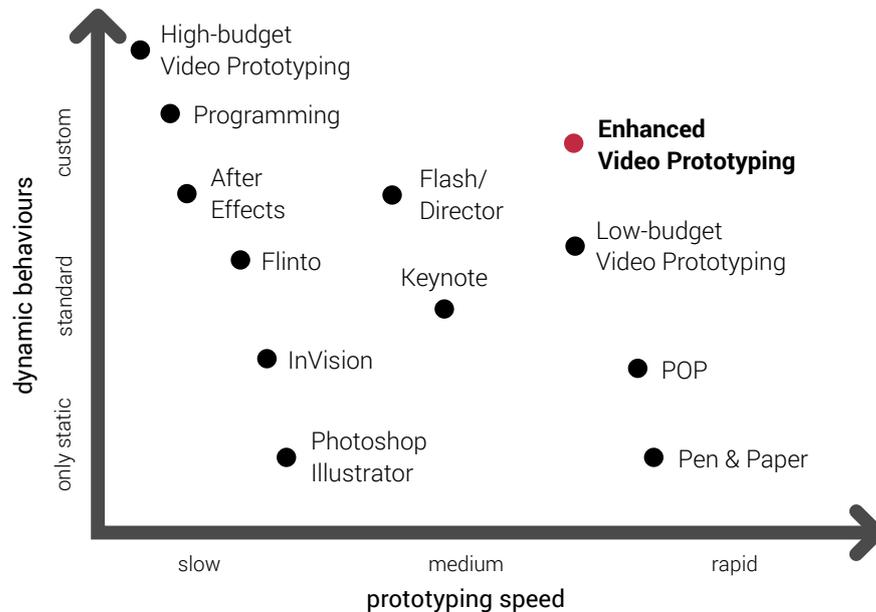


Figure 3.1: Commercial prototyping tools are ill-equipped to support rapid prototyping and highly dynamic interactions. I want to increase the speed of video prototyping while at the same time increasing its power to prototype custom and dynamic behaviors.

3.1.2 Video-based Tools

HCI research on tools to support video manipulation has different goals, such as improving video editing or video playback. For video editing, Laces (Freeman et al., 2014) lets casual video editors create video effects during capturing. Other tools focus more on professional workflows, such as rotoscoping² (Li et al., 2016), i.e. tracing to create a silhouette to extract an object from a scene and use it in another. This work aims to reduce editing as much as possible or use it only when it is relevant for exploring or communicating an interaction.

Video prototyping was introduced in the late 1980s as an effective medium for prototyping highly interactive systems (Mackay, 1988). Both, video prototyping and WOz are iterative design tech-

² For an example of the technique see the music video of "A-ha - Take on me" <https://imvdb.com/video/a-ha/take-on-me>

niques (Mackay, 1988; Mackay and Davenport, 1989). Video has been shown to be useful for a variety of activities such as user research, teleconferencing, image processing, etc. (Mackay et al., 1988).

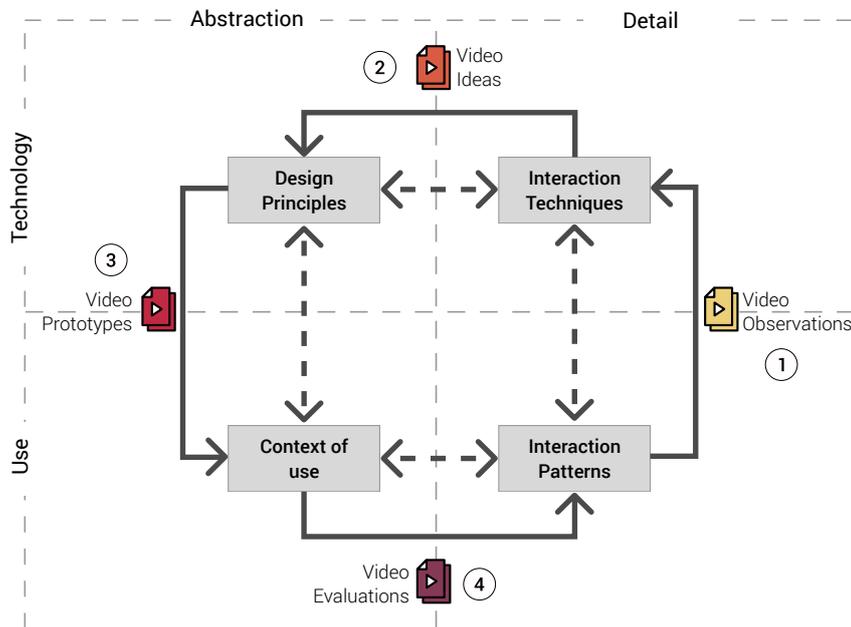


Figure 3.2: Simplified version of the design framework presented by Mackay, Ratzer, and Janecek (2000). The framework presents two main domains on the vertical axis: technology and use. The horizontal axis illustrates that design activities work on the abstractions and the details of each of these domains. Each activity has a video artifact as input and as an output. The four main considerations in the design framework are interaction techniques, design principles, context of use and interaction patterns.

Interaction designers need to navigate between the technology domain and the user domain (Figure 3.2). According to Mackay, Ratzer, and Janecek (2000), "both domains involve an inherent tension between abstractions, which ensure coherent design, and details, which ensure that the system actually works in the real world". In this design framework, the authors show how video artifacts helped a design team to integrate the abstractions and details of the technology domain and use domain. Video artifacts act as the input as well as the output of each design activity. One of the main benefits of video is that "they capture not only the basic functions of the software, but also more subtle considerations of the software as it is used in real-world contexts" (Mackay, Ratzer, and Janecek, 2000).

3.1.2.1 Video Evaluation and Observation

Designers use video as a medium to persist instances of users interacting with an already existing system, i.e. *video observations*, or a functional prototype, i.e. *video evaluations*. Video observations of user

interacting with the current system can illustrate usability problems and help designers choose the appropriate interaction techniques for the new system. On the other hand, when a design idea is materialized in a functional prototype, it is common to record the session for a posterior evaluation and to detect interaction patterns. However, recording a whole session without a clear structure generates videos that are difficult to organize and that require a time-consuming process to extract relevant information (Muller, 1991).

Researchers have presented multiple tools to aid video analysis (Mackay and Beaudouin-Lafon, 1998; Mackay and Davenport, 1989). For example, ChronoViz (Fouse et al., 2011) aggregates multiple data sources, e.g., movement, position and physiological data, with the video data under analysis to create time-coded annotations. Similarly, SceneSkim (Pavel et al., 2015) integrates other data sources such as captions, scripts and movie summaries to navigate the video.

Another approach is to use physical tokens to represent digital video clips. Video Mosaic (Mackay and Pagani, 1994) combines paper representations with digital video to layout out time-based representations in a physical space. Buur and Soendergaard (2000) also explored this idea with the *Video Card Game*: video segments are represented by physical cards that developers can use in design discussions to highlight relevant information. Sokoler and Edeholt (2002) extended this idea with *VideoCards* by adding video playback control using RFID (Radio Frequency Identification) tags on the cards.

Reflecting over the current user interaction patterns and analyzing the created prototypes in context are key activities in an iterative design process. Video can be used to persist user observations with the new prototypes or existing systems. However, in this dissertation, I am focusing on the creation rather than the analysis of video artifacts.

3.1.2.2 *Video Ideas*

According to Greenberg et al. (2012), "design is putting things in context". Video is a material that allows expressing interaction design ideas in a contextualized way. For example, video brainstorming³ (Mackay and Fayard, 1999) extends brainstorming (Osborn, 1963) with video artifacts, in order to show and act the proposed ideas. Similarly, Binder (1999) proposes that users become actors and create improvised scenarios in the context of a participatory design session, i.e. the goal is to improvise designs "on location". According to Binder (1999), video is a concrete and open medium that help designers and users to minimize vocabulary mismatches.

Video flexibility allows for alternative storytelling techniques, such as interaction relabeling and extreme characters (Djajadiningrat, Gaver, and Fres, 2000). In interaction relabelling "participants are asked to

³ for a detailed explanation see (Mackay, 2002).

consider an existing product, and, pretending that it is the product to be designed" (Djajadiningrat, Gaver, and Fres, 2000). By using extreme characters, such as designing for the Pope or a drug dealer, designers are encouraged to go beyond the average user and highlight "issues such as secrecy, status, and autonomy" (Djajadiningrat, Gaver, and Fres, 2000).

Video ideas, as well as prototypes, can have different levels of visual fidelity. In the context of a participatory design session, "quick and dirty" video ideas might suffice. However, when time and money are not a limitation, designers can use other professional movie techniques to create more elaborated "concept videos"⁴ (Wong and Mulligan, 2016). For example, Vistisen and Poulsen (2017) argue that current "corporate vision videos" might have a place in participatory design beyond been only used as persuasive or marketing artifact. Wong and Mulligan (2016) propose the use of concept videos to create design fictions where the future artifact can be embedded in a completely new contextual world or fictional reality.

In terms of storytelling, there are few tools that support the creation of rich stories without the burden of the high visual fidelity approaches. For example, Improv Remix (Freeman and Balakrishnan, 2016) is a sophisticated tool that allows improvisation in combination with digital video. The system records and reproduces video in a theatrical stage, where performers can control the system with whole-body gestures. One actor can record several improvisations that the system can combine in a final unique presentation.

Another way of reducing the cost to create these video ideas is to re-purpose existing expressive mediums such as games. For example, Machinima prototyping (Bardzell et al., 2006) consists of creating a story by reusing the characters and scenarios of an existing 3D game. Players can record audio and perform actions that represent a story within the provided virtual world. Machinima prototyping allows a quick illustration of ideas but it is constrained to the graphical elements provided by the selected game, such as the characters, items, textures, etc.

3.1.2.3 *Video Prototypes*

Video ideas are aimed at small and focused illustrations, while video prototypes tell a holistic story of the user interacting with the envisioned design to solve the identified problem. For prototyping, video becomes not only a substrate where the wizard's and user's actions are persisted but also the material to mold the prototype itself. In this dissertation, I am focusing on video as a medium to aid during the prototyping of interactive systems (Vertelney, 1989).

In the early 1990s, the Human Interface Group at Apple used video as a design tool to create future products; for them, "designing of-

⁴ For more examples see [paragraph 3.1.2.3](#)

ten meant building" (Vertelney, 1989). Vertelney (1989) differentiates between five types of video prototyping techniques: storyboards, animated drawings, cutout animation, animated objects and computer animations.

Storyboards resemble a comic book, i.e. "juxtaposed pictorial and other images in deliberate sequence, intended to convey information and/or to produce an aesthetic response on the viewer" (McCloud, 1993), but they have a different purpose and audience. Filmmakers use storyboards to outline the sequence of events in the film, the shots and the scene composition. In a similar way, designers use storyboarding to guide the recording of the video prototype (Figure 3.3).

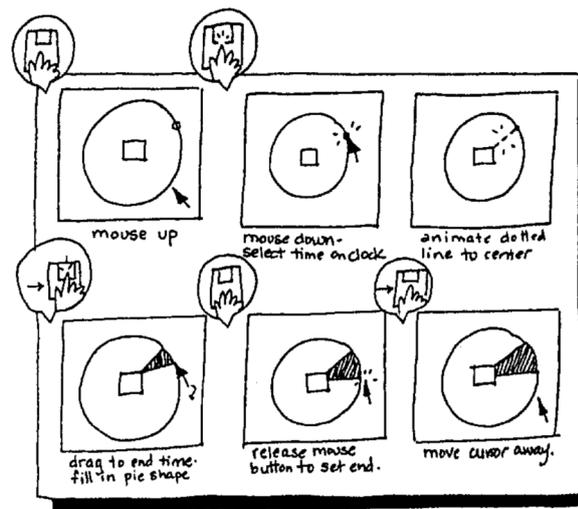


Figure 3.3: Storyboards help designers to organize the sequence of events before recording a video prototype. Here, each panel represents the computer interface. The top-left corner illustrates the user actions, i.e. mouse inputs. The text below each panel explains the user inputs and the system responses. Figure from Vertelney (1989).

STOP-MOTION TECHNIQUES To transform storyboards from static representations to dynamic ones, Vertelney (1989) mentions the use of several stop-motion techniques. For example, *animated drawings* use simple paper sketches. A background sketch is multiplied with a photocopier, and then, interface elements and system responses are drawn on top of this base sketch (Figure 3.4). The drawing represents a sequence of actions that are filmed with an overhead camera one at a time, i.e. trying to reach 24 frames per second, to transmit the effect of motion to the audience. However, this technique is extremely time-consuming.

To reduce the workload of filming one frame at a time, designers can use paper or cardboard cutouts of interface elements, e.g., to illustrate an object being dragged. The designer places the cutout on top of the background and makes it react according to the user inputs,

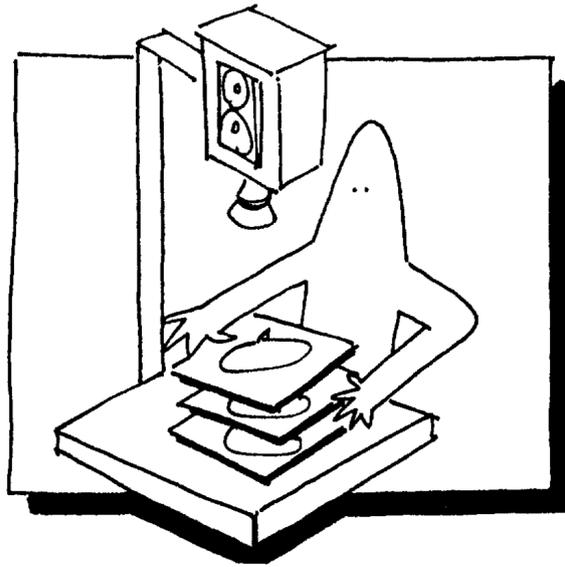


Figure 3.4: With *animated drawings*, a background is copied and reused. Sketches drawn on top of each background represent the look of the interface at short time intervals. When successive images are filmed at 24 frames per second they create the illusion of motion. Figure from Vertelney (1989).

e.g., the designer can manipulate the objects in the same way that a puppeteer moves a stick puppet. Another technique is to use an acetate paper to overlay a transparent layer over the cutouts, in order to represent elements such as the mouse's cursor moving on top of the UI (Young and Greenlee, 1992). Using cutouts reduces the burden of *pausing, filming, rearranging, resuming* by recording a continuous film sequence without interruptions (Figure 3.5). An interesting tool that supports this type of animation with paper cutouts but augmented with digital video is Video Puppetry (Barnes et al., 2008). However, this tool is targeted at artists and not interaction design, e.g., it does not allow quick iteration and the workflow emphasizes live performances. Also, this technique simplifies the simulation of movements such as translations and rotations but does not easily support deformable transformations, such as scaling a window or manipulating three-dimensional objects.

For 3D interactions, designers can create a physical model, e.g., with foam core, to prototype 3D spaces and minimize the burden of highly detailed stop-motion techniques (Figure 3.6). For example, Bonanni and Ishii (2009) propose the use of stop-motion to prototype tangible interfaces. Vertelney (1989) suggests concealing the wizard's movement of the objects by using transparent fishing lines or magnets attached to the objects and hidden from the camera. However, this technique only supports transformations that are actually possible to perform in the physical world.

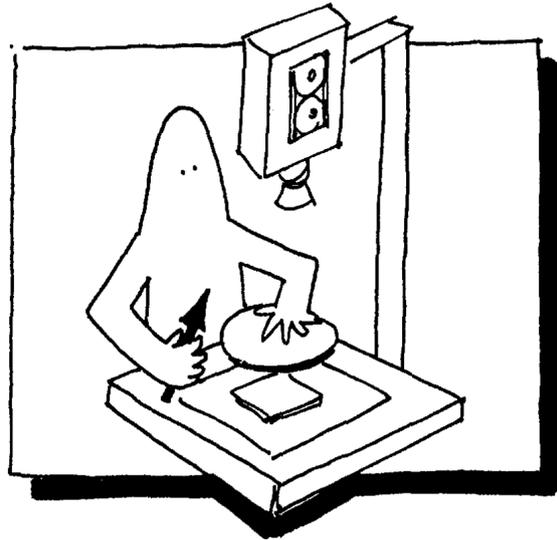


Figure 3.5: By using *cutouts* to represent movable objects, designers avoid laborious stop-motion. The cutouts represent interface elements, such as menus, or annotations in the story, such as speech bubbles. Figure from Vertelney (1989).

COMPUTER ANIMATIONS The previous techniques provide different "prototyping speeds" –some more time-demanding than others– but all had a low threshold (Myers, Hudson, and Pausch, 2000), making them easy to use. However, all of them lack mechanisms to support iteration and organization of the design, i.e. they just focus on the artifact creation and not on the design process. Making modifications to the design forces designers to re-shoot or completely change the prototype without any help from the tools. For example, a modification of the background affects all the frames using that particular background.

To overcome these iteration issues, Vertelney (1989) proposes *computer animations* and *computer scripting*. One of the first systems to create animated sketches was Genesys (Baecker, 1969). Genesys is a remarkable system that introduced many techniques for creating animations by examples using an electronic stylus⁵. For example, an animation could be created by drawing individual sketches or by making a shape follow a drawn path. Motion behaviors could be reused across sketches, and individual keyframes could be modified as well as interpolation curves.

More recent desktop animation software, such as Adobe Flash or Adobe Director (formerly Macromedia Director, formerly MacroMind VideoWorks) commoditized the creation of computer animations on personal computers. Initially, VideoWorks only supported black and white non-interactive animations, but subsequent versions added support for interactivity through the **GUI** or scripting languages. These

⁵ <https://www.youtube.com/watch?v=GYIPKLxoTcQ>

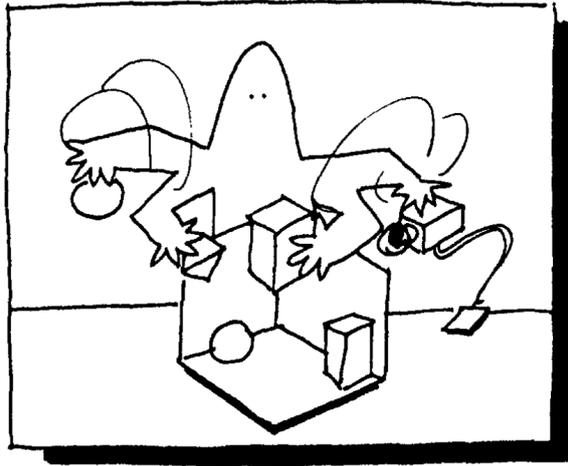


Figure 3.6: *Physical models* can represent 3D interactions. The same stop-motion techniques applied to 2D also work for 3D prototyping. The camera can even move around the physical space to prototype navigation alternatives. Figure from Vertelney (1989).

systems typically use timelines to create keyframe animations with drawings, images and text. Mainstream professional video editing software, such as Adobe After Effects or Apple Final Cut Pro, include some of these features on their video editing workflows, blurring the lines between animation and video editing (Vistisen, 2016).

Another popular tool was HyperCard (Goodman, 1998): a system for non-developers to create interactive information following the metaphor of stackable virtual cards. In HyperCard, each card contains graphics, text and buttons. User interactions on the cards, e.g., pressing a button, can bring another card to the front or execute extra actions, such as playing a sound or running a database search. Besides providing *WIMP* widgets to set up the relationships among the cards for non-programmers, HyperCard provides a scripting language, called HyperTalk, for further customizations (Listing 3.1). HyperTalk influenced ActionScript, the scripting language used in Adobe Flash. More sophisticated prototyping tools that allow the creation of animations emerged in recent years. For a review of those tools see Section 6.2.2.

In order to easily create video prototypes with custom animations, we need tools that do not require symbolic manipulations and allow a

Listing 3.1: A simple button script written in HyperTalk that reveals the next card after a mouse press

```
1 on mouseUp
2   go to next card
3 end mouseUp
```

higher prototyping speed than programming-based tools. For example, K-Sketch (Davis, Colwell, and Landay, 2008) lets amateurs create animations by drawing their motion paths. In order to animate the sketch of a rock sliding down a slope, the designers activate the recording mode and drag the sketched rock following the slope line. K-Sketch records the motion path and after pressing play, the sketch moves according to that path. To make the rock roll at the same that it slides, the designer reactivates the recording mode and rotates the rock from its center anchor. Unfortunately, K-Sketch assumes that the new recorded motion path, i.e. the roll, should override the previous one, i.e. the slide, leaving the rock rolling but not sliding. To fix this, the designer presses "Fix last motion" and the system provides three alternatives, one being roll while sliding.

I believe that this type of informal animation interfaces can be appropriate for creating video prototypes. However, few tools combine animation and video with direct manipulation. One exception is DirectPaint (Santosa et al., 2013), a system that lets illustrators sketch over video by using trajectory-based interaction techniques and optical flow⁶. DirectPaint takes a different approach than timeline-based mainstream animation tools that decouple time and space. Instead, DirectPaint combines time and space manipulation in the context of the current sketch with motion trajectories. DirectPaint uses the trajectory of the sketch as an object that can be manipulated to change animatable properties, such as position, size or opacity. While this tool is targeted at creative artists, I am interested in using similar techniques to enable interaction designers to quickly video prototype custom behaviors.

HIGH-BUDGET VIDEO PROTOTYPING While computer animations increase the ease of adding modifications to the design, they also increase the time to create the prototype. Even worse, computer animations increase the entry threshold for non-expert users by requiring programming knowledge to prototype custom behaviors.

Another alternative, particularly when the designers expect a polished visual look, is high-budget video prototyping. Two popular examples in the literature are the "The Knowledge Navigator" (Dubberly and Mitsch, 1992) and "The Starfire" (Tognazzini, 1994) videos. "The Knowledge Navigator"⁷ tells the story of a professor using a virtual agent to organize his day. The professor interacts with the agent through voice commands while the system is running on a tablet-like foldable device (Figure 3.7). Other technologies illustrated in the video include remote collaboration, shared simulations, and hypertext. The producers of the ~5 minutes movie "had about six weeks to write,

In this dissertation, I use the term video prototyping and low-budget video prototyping as synonyms unless stated differently.

⁶ a video analysis technique that generates a vector field of the apparent motion of the selected pixels between frames

⁷ <https://www.youtube.com/watch?v=mE2Z30pyw8c>

shoot, and edit the video—and a budget of about \$60,000" (Dubberly and Mitsch, 1992). It is important to note that "these pieces were marketing materials": the intention was not to create a new interface but to increase computer sales "by suggesting that a company making them has a plan for the future" (Dubberly and Mitsch, 1992).

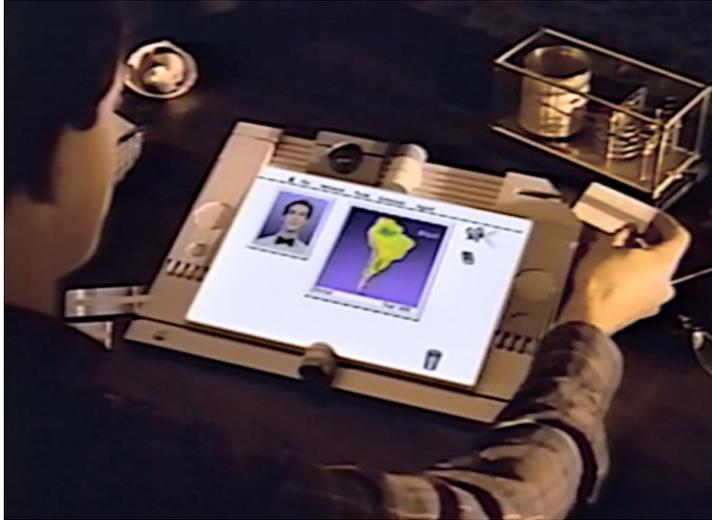


Figure 3.7: A video prototyping of a professor interacting with a virtual agent. The professor requested the agent to copy information about "deforestation in the Amazon" in the physical card-storage device. From "The Knowledge Navigator" by Apple Inc.

"The Starfire"⁸ video prototype illustrates an "integrated computing-communication interface" (Tognazzini, 1994). Designers turned to high-budget video prototyping, instead of just computer animations, because they wanted to show the experience of using the system. "The Starfire" is mainly shown in a "curved desktop display" that has a blue-screen replaced with the interface in post-production (Figure 3.8).

Tognazzini (1994) wanted to highlight not only how well the system worked but also how to recover from unexpected situations. For example, in "The Knowledge Navigator" the intelligent agent never misinterpreted the voice commands of the professor. "The Starfire" included instances of unrecognized vocal commands and unintended actions, such as scanning a sandwich laying on top of the desk. Tognazzini (1994) makes several observations about high-budget video prototyping:

1. *"Interaction techniques most easily accomplished on film may be difficult or even impossible to actually implement on the computer."*
2. *"The actors in a video prototype will show no distress in using the interface, regardless of its quality."*

8 <https://www.asktog.com/starfire/>



Figure 3.8: The prototype of an interactive curved desktop display. Placing objects on the flat area allows the system to scan the contact area, e.g., to scan a document. The workstation can be controlled with a mouse or via voice commands. From "The Starfire" by Sun Microsystem Inc.

3. *"In real life, things do not always work as planned. In film, they will unless you are very careful."*
4. *"Hardware in video prototypes can be complex to the point of impossibility and still appear to be easy to fabricate."*
5. *"In the film medium, the simpler and less direct the physical interaction with the interface is, the less expensive it is to film it."*
6. *"The first goal in a video prototype is to communicate a vision to the viewing audience. The second goal is to design a usable system."*
7. *"Video prototyping offers the opportunity to explore social, as well as technical issues."*

These types of video prototypes require a massive amount of planning. Multiple locations, actors and visual effects are needed to communicate all the details of an interactive system within a story with high visual requirements. In the case of "The Starfire", the video included locations such as an airport with dozens of extras, an outdoor scene with a car filmed with a professional camera crane, and a conference room with a video conferencing system based on holograms.

Some researchers encourage high-budget techniques to emphasize the context of use in the prototype. For example, Bardram et al. (2002) and Halskov and Nielsen (2006) propose using *virtual video artifacts* created in a recording studio (Figure 3.9) with blue screens and theatrical props to mix physical and digital 3D objects.



Figure 3.9: The Virtual Studio at CAVI (Centre for Advanced Visualization and Interaction) in Aarhus, Denmark.

Halskov and Nielsen (2006) say that a "general estimate of the resources put into the various productions is as follows: first, creation of the story line and the 3D models, 10 days; second, production in studio and on location, 8 days; last, post-production including voice over: 2 days". This is a lengthy process when compared with rapid prototyping techniques. However, once the virtual video prototype is created, modifying the prototype might be faster than with rapid prototyping, particularly when dealing with large physical objects.

Another problem with these "virtual" techniques is the decoupling between actual experience and acting. In some cases, actors interact with a blue screen that does not respond to the user inputs during recording, i.e. the interface is embedded in post-production. However, using a monitor with a preview of the final composition helped participants. Thanks to this monitor, the actors had a sense of the prototyped environment and remembered being on the envisioned set rather than in the recording studio (Halskov and Nielsen, 2006).

These high-budget techniques require bulky equipment and large studios. However, the advent of mobile interaction allowed video tools to move from the studio and into the wild (Jokela, Karukka, and Mäkelä, 2007). For example, Motif (Kim et al., 2015) is a mobile storytelling tool that combines story patterns from curated videos to facilitate the creation of stories by amateur users. Mobile phones also opened the door to inexpensive ways of exploring new technologies, such as AR. For example, Sá et al. (2011) describe an exploratory experiment to video prototype an AR experience, a location-based social application called The Friend Radar. Most participants that watched the 30 seconds video prototype "thought that it was easy or very easy to understand the concept" (Sá et al., 2011). Berning et al.

(2013) presented an AR tool that extends the cameras of mobile devices with a lens to record panoramic video. The recorded 360° videos can be edited and played on the mobile device (Berning et al., 2013).

In summary, there are two main approaches to video prototyping: low- and high-budget. Low-budget techniques, i.e. rapid video prototyping, often lack the means to prototype non-idiomatic interactions. Stop-motion techniques paired with digital video editors, such as Adobe Premiere or Final Cut Pro, provide an interesting balance between low- and high-budget techniques (Löwgren and Jonas, 2004). However, mainstream video editors are not targeted at interaction design and stop-motion is a laborious and time-consuming process. Digital tools, such as scripting languages and computer-based animations, help designers prototype dynamic interactions, i.e. they increase the ceiling; however, they also demand designers with programming skills or vast expertise with these tools, i.e. they increase the entry threshold. The more realistic techniques, such as high-budget video prototyping, are only implementable with the right equipment and they generally require the intervention of professional videographers and editors. Also, these realistic approaches usually serve a more persuasive role rather than an exploratory one, diminishing the incentives to generate multiple alternatives.

Dhillon et al. (2011) conducted a case study comparing video prototypes with low and high visual fidelity. Their results suggest that there are no differences between these video prototyping styles "regarding the amount or quality of feedback one should expect from a low or a high visual fidelity video" (Dhillon et al., 2011). For this reason, I want to create video prototyping systems that support dynamic interactions but maintain the sketchiness of early-stage techniques to encourage exploration.

3.1.3 Video Prototyping Systems

There are few examples in the literature of video prototyping systems targeted explicitly at interaction design. miniStudio (Kim, Kim, and Nam, 2016) is a prototyping toolkit that lets designers explore ubiquitous computing spaces (Figure 3.10). A camera-projector system detects hidden markers on small paper figures and projects digital images on top of them. Similarly to the physical models shown in Figure 3.6, designers manipulate the miniature physical model to modify the layout. Each tracked element has an identification that the system uses to trigger certain events, such as location, distance, motion and orientation. The system is well suited for designers because it is integrated into Adobe Photoshop. System visual outputs are coordinated with the programmatic events by following an ad-hoc naming convention in the exported graphics folders and files. The manipulation space, i.e. the miniature physical model, and the creation

"When the design situation handles technologies or interactions with few or no conventions the design situation becomes non-idiomatic, and design idioms often become insufficient." (Vistisen, 2016)

space, i.e. the software on the desktop computer, are decoupled, thus generating indirections and extra work not suitable for a "quick and dirty" process.

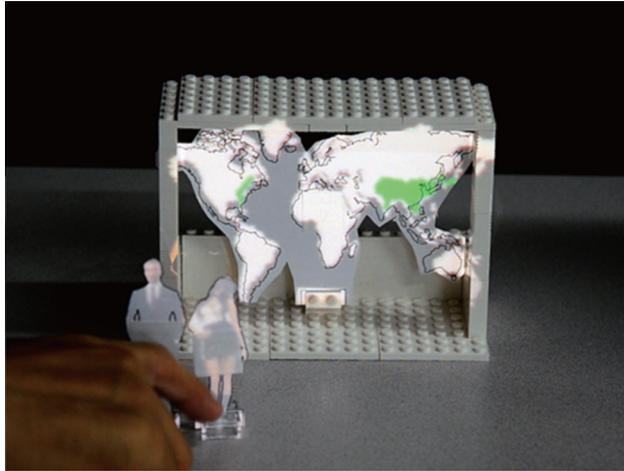


Figure 3.10: The miniStudio system uses projections to prototype proxemic interaction. Figure from Kim, Kim, and Nam (2016).

My goals are similar to those of DART (MacIntyre et al., 2004), i.e. supporting early design stages and recording synchronized data. DART is a rapid prototyping system that overlays graphics on top of the user's view by using a see-through display. The DART system consists of a plugin for Macromedia Director and preloaded code snippets, i.e. Director behaviors. The plugin integrates the information from the cameras, the trackers and other sensors. The default behaviors can be dragged onto the sprites in the scene but new behaviors require additional coding. Also, the AR experience can be extended by adding multimedia content, such as video or audio. However, DART assumes that the designers are expert in Director. In a later study "interviewees consistently expressed a desire for a tool to support prototyping without coding" (Gandy and MacIntyre, 2014). Also, DART is focused on AR interactions but I want to support multiple interaction styles without requiring coding skills in an early-stage process.

Finally, Remote Paper Prototype Testing or RPPT (Chen and Zhang, 2015) is used to run live testing sessions with real users. A wizard records a paper prototype with a smartphone camera positioned above the surface, e.g., a smartphone attached to a bookstand. This video feed is streamed with an existing video call software, e.g., Google Hangouts, to the tester's device. The tester interacts with this video feed as if it were an interactive UI. At the same time, the tester wears smartglasses with a camera, e.g., Google Glass, that stream back to the wizard the user's view and its location. With this information, the wizard can give instructions to the tester and update the paper prototype accordingly. Like RPPT, I believe that live streaming of paper prototypes is an interesting approach for combining physical

and digital artifacts. However, RPPT does not let designers iterate over the recorded video because it is targeted at testing rather than design. By contrast, I want to help designers create reusable video prototypes and explore alternatives.

In summary, these systems propose interesting mechanisms but they either require complex setup or they do not easily support the modification of the original idea being prototyped. I believe that video prototyping tools should help interaction designers organize the video session by themselves and manipulate the video material to create non-idiomatic interactions.

3.2 HOW DO INTERACTION DESIGNERS USE VIDEO TODAY?

This section presents work done in collaboration with Lai Linghua and Wendy Mackay

I was introduced to the video prototyping technique during my master program in Human-Computer Interaction and Design at Université Paris-Sud. Wendy Mackay taught a course on Designing Interactive Systems and the final assignment was the presentation of a video prototype. After this hands-on experience, it was clear to me that video is a powerful medium for prototyping. However, I had worked in software companies and design agencies before and I had never seen video used as a rapid prototyping technique. Was this a general phenomenon? If so, what are the barriers that designers encounter when using video for prototyping?

3.2.1 Method

To answer the previous questions we decided to conduct a questionnaire study. We decided to distribute an online questionnaire to collect information about current and professional uses of video for prototyping from interaction design students, professionals and HCI researchers.

3.2.1.1 Participants

We distributed the questionnaire through our professional networks, the university communication channels, and public organizations such as the Interaction Design Association⁹. In total, we collected responses from 34 participants in 11 countries. Most participants have between 1 and 3 years of design experience (Figure 3.11).

3.2.1.2 Procedure

The questionnaire (see Appendix Section A.1) focused on how interaction designers and HCI researchers prototype interactive systems during an early-stage design process. The questions follow the critical incident interview technique, i.e. concrete questions about recent

⁹ <https://www.linkedin.com/groups/3754>

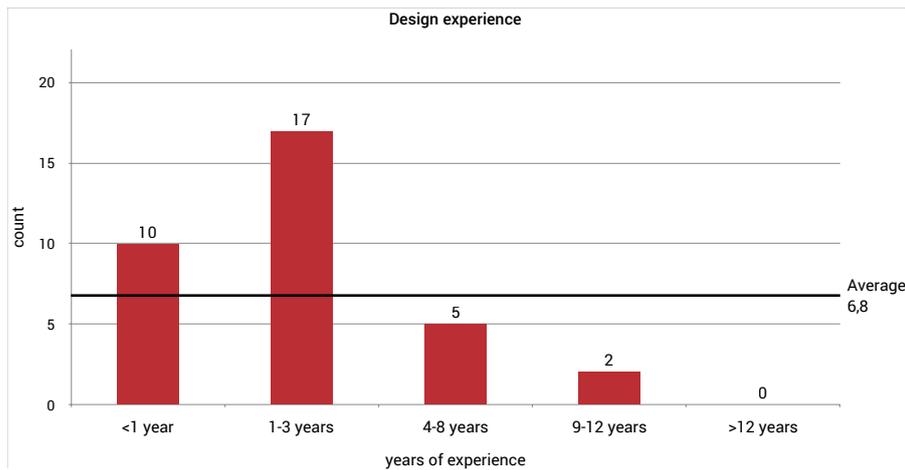


Figure 3.11: Distribution of the participants' design experience. Half had 1-3 years of experience. The other half: 29% less than one year, 15% between 4-8 years and 6% between 9-12 years. The average experience is 6.8 years.

events are prioritized (Mackay, 2002). Towards the end of the questionnaire, more open questions enable us to collect richer answers.

3.2.1.3 Data Collection

We distributed the questionnaire online and the answers were automatically saved on an online spreadsheet file.

3.2.2 Results and Discussion

Participants' work environments are distributed between academic research in HCI (20 cases) and industry (17 cases), i.e. freelancers, startups, design firms, and large companies. Some participants had more than one job, e.g., being a researcher and a freelancer or working for a startup and a design firm.

We asked participants to reflect on the amount and type of artifacts used during their most recent project (Figure 3.12). Our results echo the findings of Carter and Hundhausen (2010): the most widely used tools are hand-drawn sketches. All participants but one (97%) said that they used at least one hand-drawn sketch in their project and 30% used more than ten. However, and as a close second place, we found that computer-drawn sketches are widely used as well. 97% of the participants said that they used at least one computer-drawn sketch on their project but only 13% said that they used more than ten computer-drawn artifacts. The least used design artifacts were video illustrations (10%), followed by computer animations (9%).

We were also interested in how much time did designers spend creating these artifacts (Figure 3.13). Only 6% spent more than half of the design process creating hand-drawn sketches. This is not surprising due to its value as a rapid prototyping technique. However, 41% spent

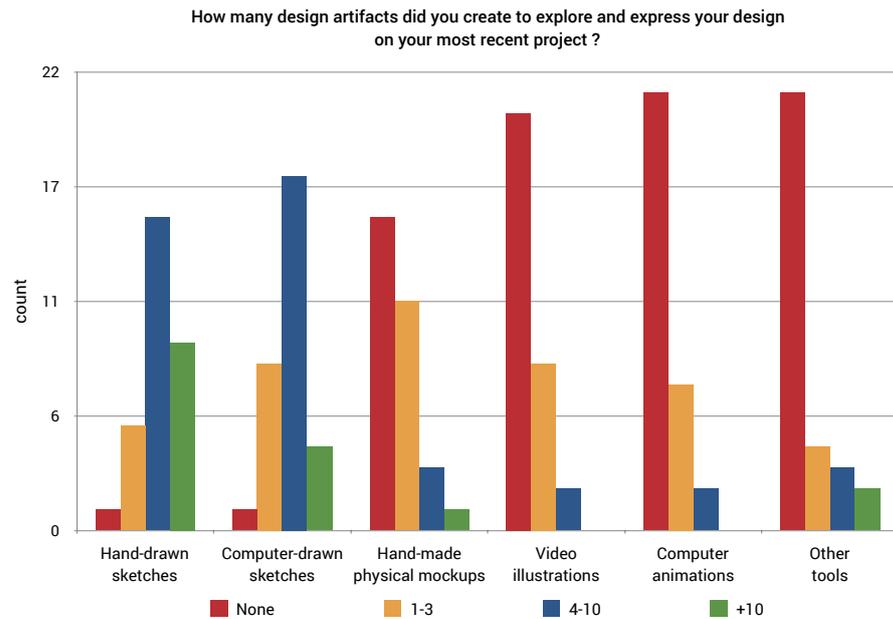


Figure 3.12: Amount of design artifacts from the most recent project. Hand-drawn and computer-drawn sketches are among the most created artifacts. The least created are video illustrations and computer animations accompanied by a mix of tools, such as software mockups –Android and HTML5–, prototyping tools –InVision–, and presentation software –Google Slides–.

half or more of the design process creating computer-drawn artifacts. Two participants reported spending more than 80% of the design time creating static digital drawings.

Not many participants reported the time spent when creating computer animations and video illustrations. However, from the 12 participants that reported their time spent while creating video illustration: 4 spent 0-20%, 3 spent 20%-40%, 4 spent 40%-60% and 1 spent 60%-80%. One participant reported spending more than 80% of the design time creating computer animations.

The answers were varied when we asked about how they used these artifacts to explore designs. However, most participants reported that the most time-consuming activities were the creation of refined computerized design artifacts:

Most of the time I have spent with high fidelity prototypes

Most participants described a simple progression from hand-drawn sketches to computer-drawn sketches, while others tried to avoid digital sketches:

I start with hand-drawn sketches, to quickly ideate what the prototype will look like and how it will act. Then I usually translate that directly to the high definition prototype. I rarely do computer drawn sketches - mostly when I'm not sure about

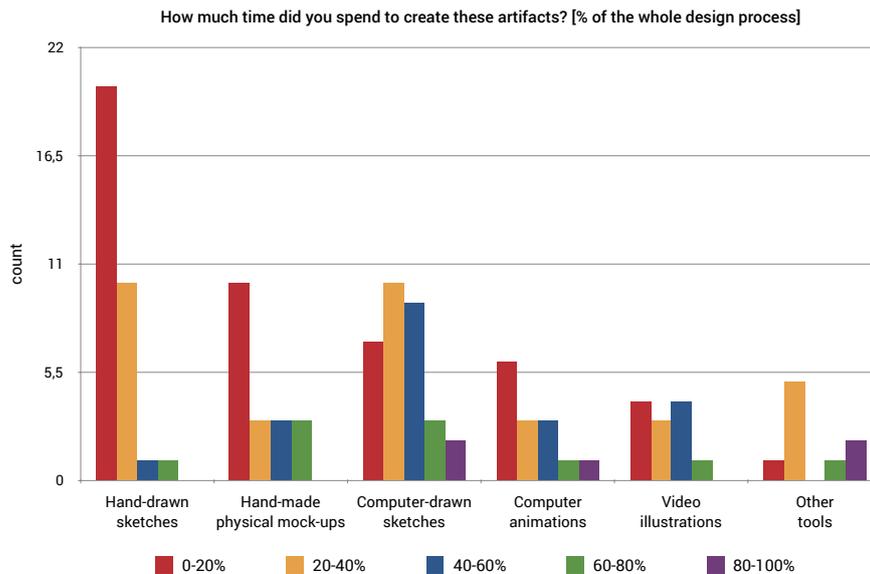


Figure 3.13: How much time was spent creating these design artifact? For 59% of the participants, the creation of hand-drawn sketches only took between 0-20% of the process. Most participants said that computer-drawn sketches took between 0-60%. However, one participant responded that creating the computer-drawn artifacts took between 80%-100% of the time. Another participant reported that video and computer animations took between 60%-80% of the process.

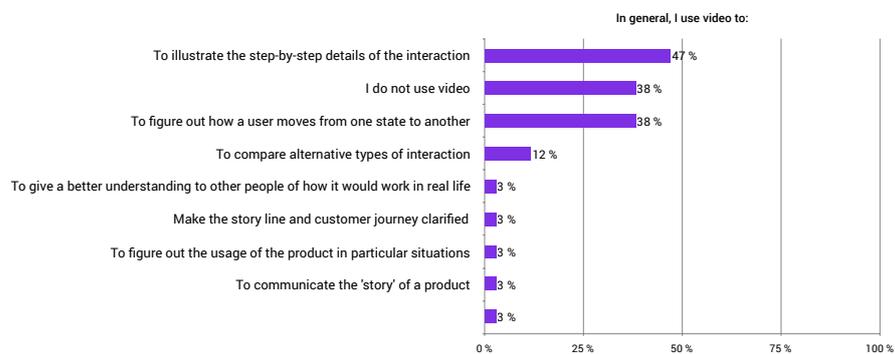
the interaction, I would prototype 3-10 iterations of the same module, until I get it right. This is either in computer-drawn sketches and wireframes or in high definition prototypes.

In terms of video usage as an early-stage tool, a participant said:

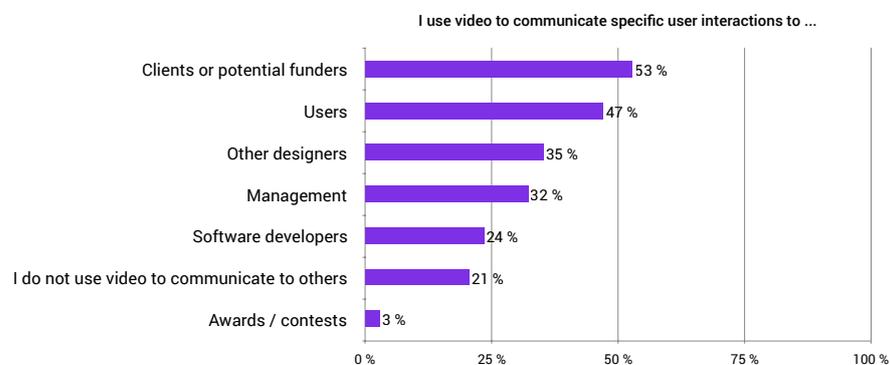
In the beginning, when brainstorming about the idea, I used papers and pen. And then we wanted to make a paper prototype (videotaped) so we needed to make the physical mock-ups. I needed to make the computer-drawn and animations when I had to present the idea to other people in a semi-formal presentation. That took quite long to make. And finally, I started coding the application prototype, from which I could already make the video illustration. That took the longest, of course, but once I had it it's easier for me to keep working on it so it evolved into the real implementation rather than just a prototype.

We asked participants about their use of video. In general and beyond their most recent project, 62% of the participants (21/34) used video for interaction design (Figure 3.14a). Most of them used video to illustrate the step-by-step details of the interaction (47%) and to figure out how a user moves from one state to another (38%).

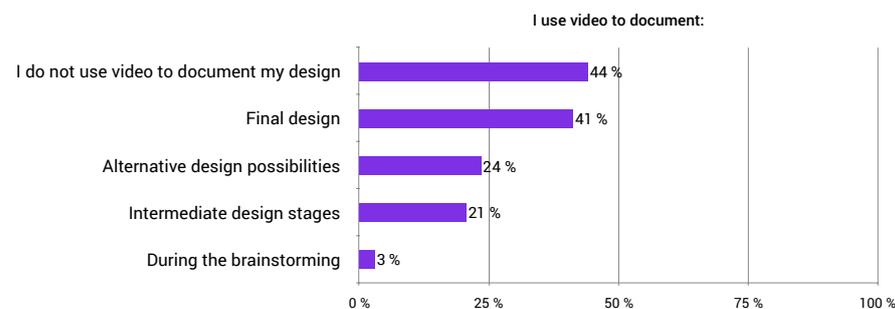
We also asked if they used video to communicate specific user interactions to different stakeholders (Figure 3.14b). Only 21% of the



(a) 47% of the participants (15/34) use video to illustrate details of the interaction and 38% (13/34) to figure out how to transition from one state to another.



(b) 79% of the participants (27/34) communicate specific user interactions through video with clients, potential funders, and users.



(c) 44% of the participants (15/34) do not use video to document designs. Of the rest, 41% (14/34) use video to document the final design.

Figure 3.14: We asked participants about their use of video as a design (Figure 3.14a), communication (Figure 3.14b) and documentation tool (Figure 3.14c).

participants (7/34) do not use video to communicate interaction to others. The vast majority used video to communicate with the clients (53%) and final users (47%) and, to a lesser extent, to communicate with other designers (35%), managers (32%) and software developers (24%).

Additionally, we asked if they used video as a documentation artifact (Figure 3.14c). 44% of the participants (15/34) do not use video for documentation. From those who did, most documented the final design (44%), but only a few documented alternative design (24%) or intermediate designs (21%). It is surprising to see that only one participant explicitly mentioned using video as an ideation tool.

Finally, we asked about the barriers to using video, in order to understand why the majority of the participants (67%) did not use it as a design tool (Figure 3.15). They two main barriers are the lack of time or resources to edit video (76%) and to record video (68%). Other barriers included the difficulty of finding relevant video clips (21%) and the lack of access to video equipment (21%). Only 15% of the participants (5/34) did not find video useful for exploring or expressing the design.

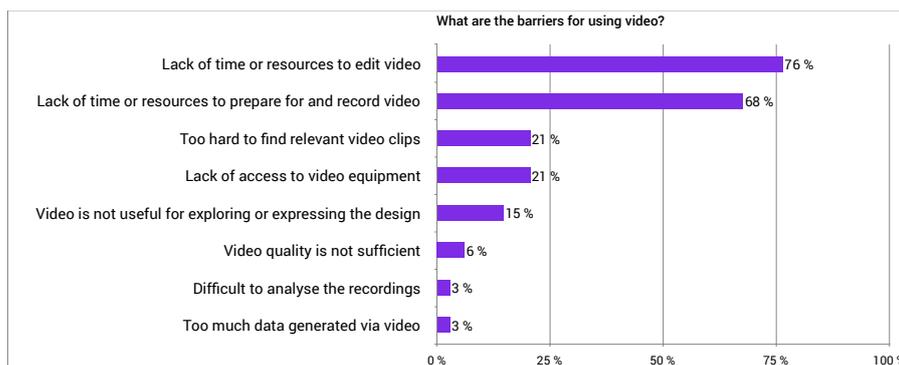


Figure 3.15: The main barriers to using video are the lack of time or resources to edit (76%) and the lack of time or resources to prepare for and record video (68%).

Many participants believe that having dynamic artifacts is critical to explore and refine interaction design ideas, yet few chose video as an early-stage tool. Due to the cost of creating a video artifact, most participants only use video to document the final design rather than to explore early-stage ideas. In terms of interaction prototyping, participants are forced to learn complex prototyping tools or directly code interactive prototypes, due to the barriers to using video.

This study helps to identify the main barriers to using video, i.e. recording and composing video is a time- and resource-intensive task. In the following chapters, I present two video prototyping tools created specifically to reduce these two barriers in early-stage prototyping.

VIDEOCLIPPER: PLANNING VIDEO CAPTURE ON AN INTERACTIVE STORYBOARD

First, I briefly illustrate how a typical video prototyping session unfolds with a concrete example. Afterwards, I introduce VIDEOCLIPPER as a tool to support planning in video prototyping.

This chapter presents work done in collaboration with Wendy Mackay.

4.1 A TYPICAL VIDEO PROTOTYPING SESSION

Initially, an ideation session usually precedes the video prototyping. Let us consider a group of designers working together. The starting point could be a problem or need from a potential user group, e.g., organizing a night out with some friends. In this case, the designers follow a Participatory Design methodology (Muller and Kuhn, 1993). Mackay (2002) proposes interviewing target users about their particular problem domain to gather more information. The goal is to understand more about the problem that the user is facing. Osterwalder et al. (2014) proposes distilling the *job to be done*, i.e. what is the user trying to achieve?, the *pains*, i.e. what is annoying or troubling for the user?, and the *gains*, i.e. what would make the user happier? For example, for organizing a night out with friends the *job to be done* is deciding which locations and activities are part of the itinerary. However, satisfying the preferences of every friend might be time-consuming and *painfully* stressful. Finally, users could *gain* some time by finding more efficient ways to create the itinerary.

This example is based on a real group of students creating a video prototype in the context of a design course

After this analysis, the designers have a better picture of the problem to solve. Now, they can start proposing potential solutions. In order to generate ideas, they decide to do a brainstorming (Osborn, 1963). Brainstorming is a fairly popular technique with two basic goals: defer judgment and aim for quantity rather than quality. The designers present several ideas and rank them with a simple majority vote. Once the team agrees on which ideas should be pursued, they need to explore them in more detail. This is referred to as "sketching" (Buxton, 2007) or prototyping for exploration (Floyd, 1984). They decide to create a video prototype to explore the five selected ideas.

The team needs to define a scenario (Carroll, 1995), create a storyboard (Lelie, 2006) and record the prototype in context. A scenario contextualizes the user's activities in a real-world setting (Mackay, 2002), it identifies the user, e.g., by using *Personas* (Cooper, 1995), and makes explicit the sequence of actions to be performed. For example, the scenario can describe a group of friends deciding where to go while they are walking outside or while they are communicating through

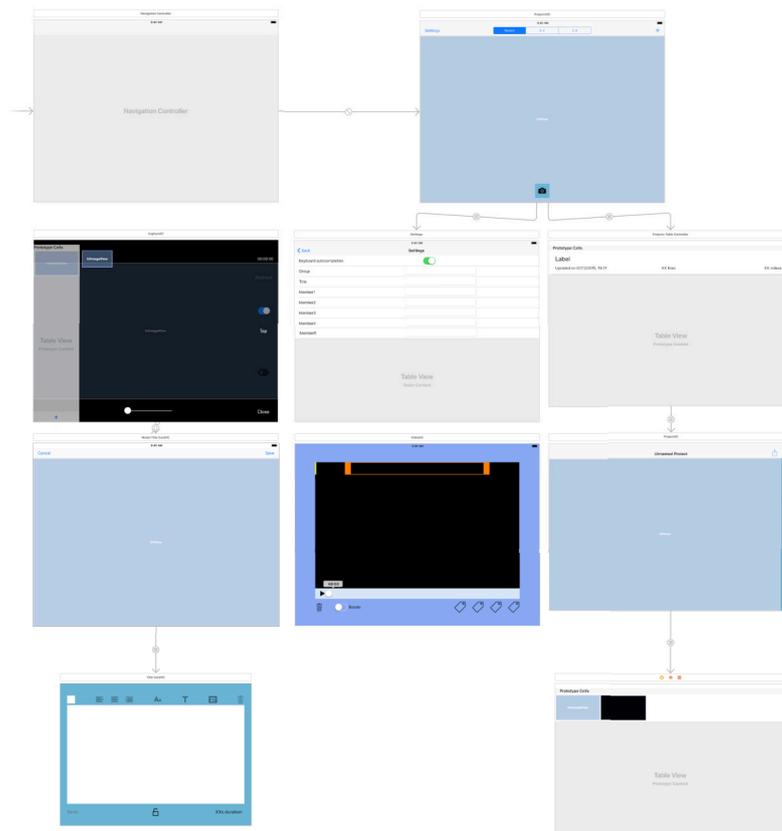


Figure 4.1: A digital UI storyboard created in Apple Xcode, focused on the user interface layout and the flow between screens.

social media. The scenario can also illustrate breakdowns (Mackay, 2002). Social breakdowns, such as a friend disliking a location or being allergic to a certain food, or technical breakdowns, such as having problems with the Internet connection or running out of battery. Adding these edge cases in the scenario pushes the design to handle these situations.

The storyboard is a materialization of the proposed scenario, i.e. a blueprint of the video that the team is about to shoot. There are different storyboarding techniques, some focused solely on the user interface (Figure 4.1) while others, as in film-making, are more focused on the narrative and the overall story (Figure 4.2). In a rapid video prototyping session, the storyboard is usually drawn by hand, contains sketches that illustrate the scene and text descriptions working as the movie script. In our example, one idea was to create an interactive and collaborative visualization with the potential activities and the constraints of every friend, in order to facilitate the organization of the itinerary.

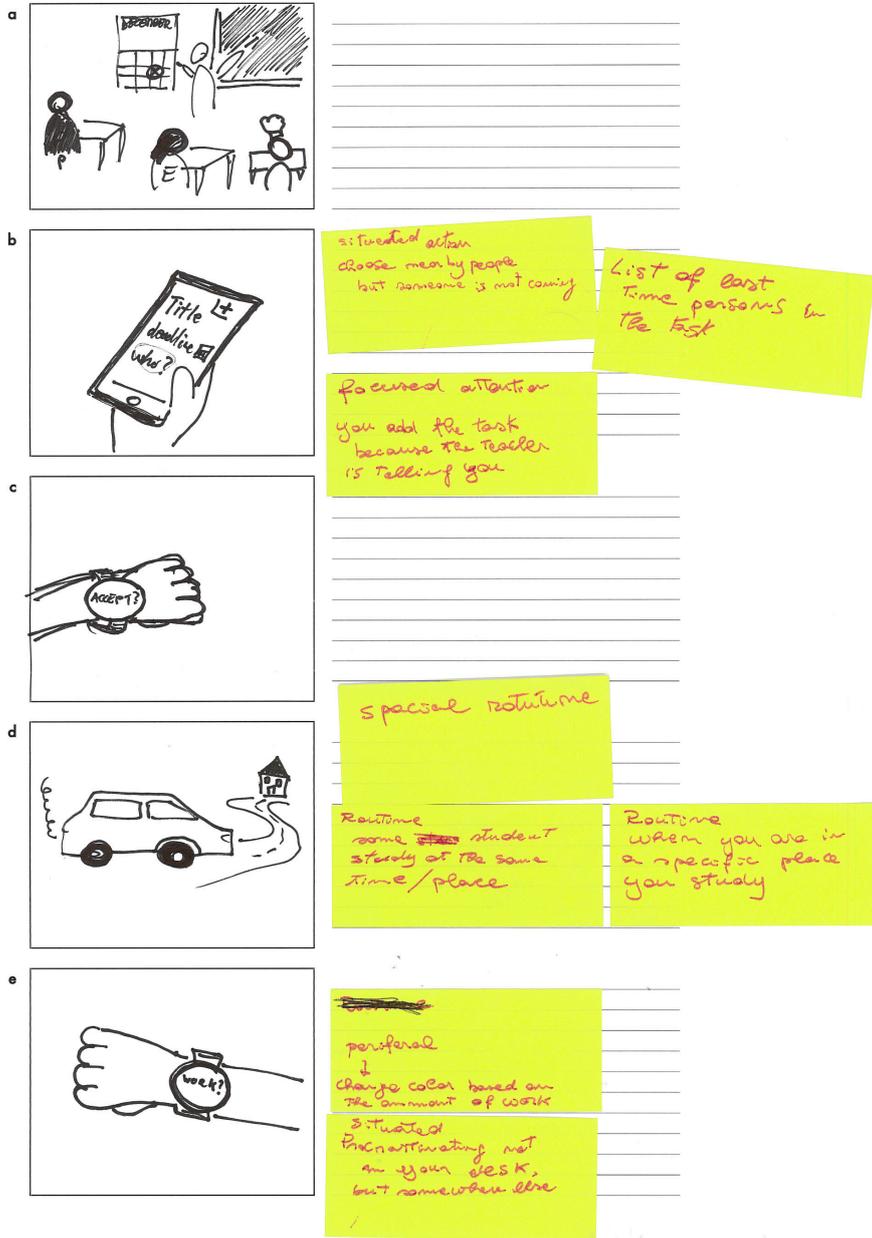


Figure 4.2: A narrative hand-drawn storyboard that combines sketches with textual descriptions and sticky notes.

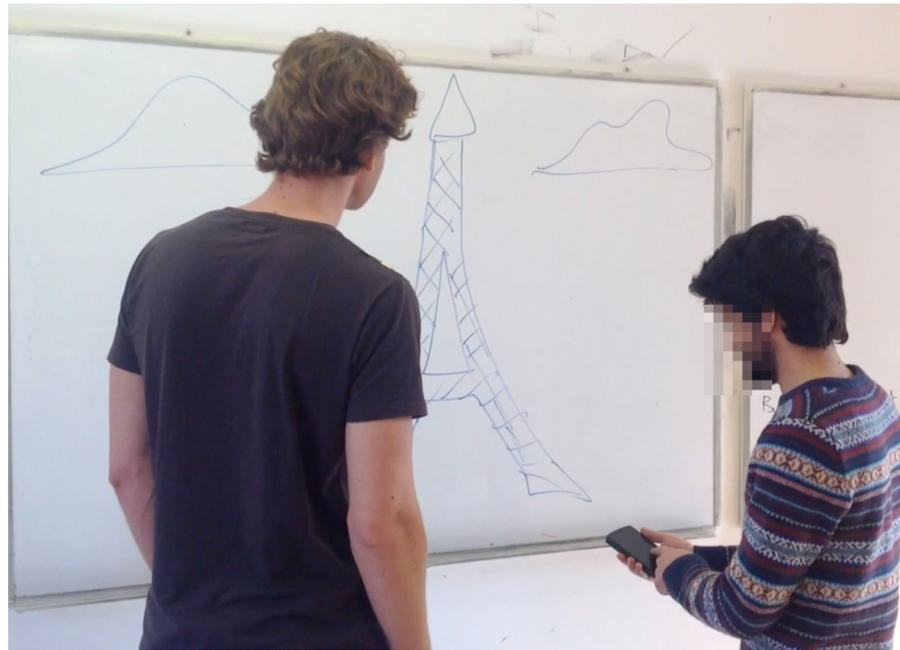
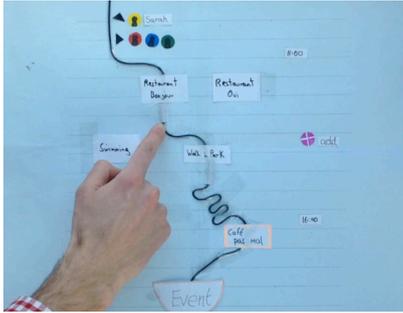


Figure 4.3: Designers improvise a background of the Eiffel Tower by sketching over a whiteboard.

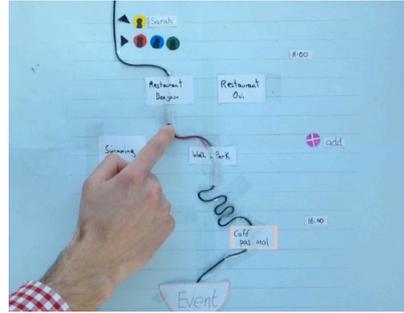
Finally, the team starts shooting the video. Designers can record with a professional camera or with a mobile device. Whichever recording device is used, it is important to have a firm stand, such as a tripod or a chair to facilitate re-shooting a similar clip if needed. Designers use art supplies to illustrate the envisioned idea, such as pen and paper, sticky notes, markers, etc. They also need to create the scenography for the video according to the script depicted in the storyboard. Sometimes it is easy to prepare the "filming set", e.g., when the current location is the final environment, such as being at the office. Other times designers need to improvise some props, e.g., designers can draw an Eiffel Tower on a whiteboard to create a "quick and dirty" background to illustrate a touristic environment (Figure 4.3).

During recording, it is common to see designers facing unexpected situations. For example, the designers decided to prototype an interactive graph diagram, where each node is an event and the edges represent, from top to bottom, an ordered sequence of activities. However, one of the brainstormed ideas was to have edges that "behave like a rope" instead of straight lines. They decide to illustrate these "curvy connections" by re-purposing the cable cord of one of the designer's earphones (Figure 4.4a).

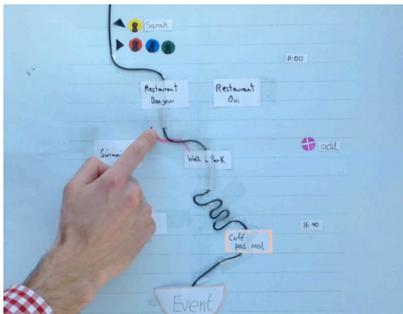
Designers also wanted to create a sibling relationship between events by dragging these curvy edges. This might sound too detailed for an early-stage prototype but the designers really care about the *feel* of the interface, i.e., its interactivity. The owner of the earphones was not happy about cutting the cable to mimic this behavior. Instead,



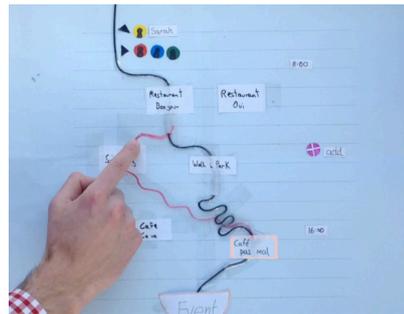
(a) Designers re-purpose an earphone's cord to prototype the curvy edges.



(b) A transparent paper with a drawing is positioned below the user's hands.



(c) Transparent paper is moved to illustrate the dragging feedback.



(d) The same technique with a mirrored drawing is used to show the final result.

Figure 4.4: The designers use transparent paper to mimic the dragging feedback on top of the paper prototype.

the designers decided to use a transparent paper to draw a red line representing the segment of the edge and drag it on top of the interface. To make the dragging feedback appear, designers pause the recording as soon as the actor starts to drag the selected edge. The actor keeps her hands in this position while the other designers position the transparent paper with the red edge accordingly, under the user's hand (Figure 4.4b). Misalignments with the previous shot can create an abrupt transition and holding the same position for a long time can also become a burden for the actor. To show the dragging feedback, the actor moves the transparent paper alongside her finger (Figure 4.4c). Designers rely on the same technique to illustrate other changes. For example, when the actor drags the edge far away from the origin node, the edge changes its shape, e.g., in this case the drawing is mirrored (Figure 4.4d).

We just described a simple video prototyping process for one of the ideas selected after the brainstorming. During the shooting of the remaining ideas, designers can find scenes that should share the same background or that have segments of clips similar to previously recorded scenes. Because they are filming with the camera application of a smartphone, there is no easy way to reuse these props and scenes without post-production video editing. Even more, current

video tools generally see video as a linear stream of data. In order to present or share their final video prototype, they need to import all their video files into a video editing tool, such as Apple iMovie or Adobe After Effects. However, these tools organize the video clips as a unidimensional stream, which hinders the reorganization of the clips by their semantic purpose, e.g., repositioning a group of clips introducing the Personas or showing multiple alternatives to recover from the same technical breakdown. We believe that videos should be grouped "as you shoot", to help designers quickly find and organize the scenes without a context switch to a separate and complex editing software.

4.2 DESIGN GOALS

In our survey (Section 3.2), the main barriers for video usage in interaction design are the time-consuming activities of recording and editing the material. However, the video prototyping process already features solutions to these problems. The goal is to materialize a design in the context of a concrete scenario instead of creating a polished movie aimed at persuading the audience into buying a product. As illustrated in Section 4.1, capturing during video prototyping should happen "on the go" and video editing should be avoided as much as possible (Mackay, 2002). Unfortunately, mainstream video tool interfaces preclude designers from following these simple guidelines. Video prototyping exists as a methodology but there is a lack of tooling.

We want to provide a video tool specifically designed to support a rapid and mobile video prototyping process. Since the most popular camera is now a smartphone¹, this tool should be compatible with mobile devices. However, a design team will have difficulty manipulating multiple videos on a smartphone's small screen. For this reason, we think that a tablet device, such as an iPad, provides a good compromise between mobility and screen real estate.

From previous experience, we observed that designers are generally more efficient during video capturing when they start with a concrete storyboard. However, slightly modifying the storyboard while shooting is not uncommon, e.g., inconsistencies are detected, new ideas pop-up, and others are discarded. Pen-and-paper is the default medium for storyboarding, however, when the final video changes this specification is rarely updated. We think that a digital storyboard can complement the shortcomings of the paper-based version.

Finally, we want designers to connect shots without complex video editing, i.e. we want them to use *straight cutting*. However, designers usually want to make objects appear/disappear or move according to

¹ iPhone takes top rank as Flickr's most used camera (2017) <https://www.cnet.com/news/iphone-top-camera-flickr-2017-report/>

direct or indirect user interactions. In film-making, creators use techniques such as "invisible cuts" (Bolchini, Pulido, and Faiola, 2009) or stop-motion (Shaw, 2017) to create these simple effects. In an "invisible cut"², shots with slightly different content are edited to look continuous, in such a way that objects can magically appear, disappear or change. In stop-motion³, pictures with small increments of the action are recorded sequentially, i.e. frame-by-frame; when the sequence is played back at a regular speed, e.g., 24 frames per second, the illusion of fluid motion is created. However, these techniques are prohibitively laborious for rapid video prototyping.

In summary, current tools encourage designers to capture too much video and waste time editing it later. We want to take advantage of the capabilities of digital cameras by providing a structure that supports disciplined video capture, limits editing, and facilitates reuse.

4.3 VIDEO PROTOTYPING WITH VIDEOCLIPPER

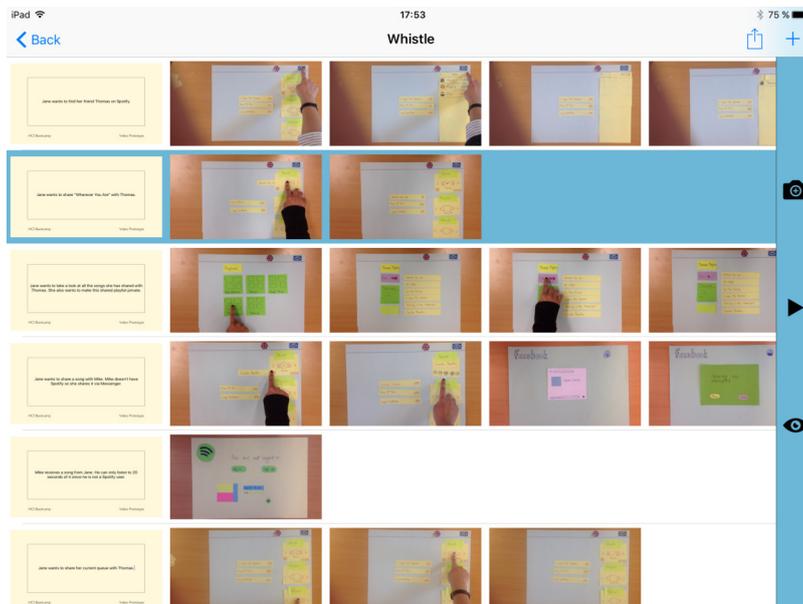


Figure 4.5: The VIDEOCLIPPER's storyboard view organizes each video prototype as collection of ordered Lines. Each Line starts with a TitleCard followed by the corresponding video clips. Designers can open the capture view by pressing the camera icon on the right.

VIDEOCLIPPER is a mobile, tablet-based tool that supports both planning and last-minute video capturing to create a finished, reusable video artifact at the end of a design session. With VIDEOCLIPPER each video prototyping session is organized as a two-dimensional

2 Zach King - Back to school problems (2016) <https://www.youtube.com/watch?v=ApnBw--188>

3 Stop Motion in King Kong (2005) <http://www.criticalcommons.org/Members/pcote/clips/king-kong-stop-motion.mov/view>

interactive storyboard (Figure 4.5). In the same way that a presentation software, such as Keynote or PowerPoint, organizes the content into slides, VIDEOCLIPPER organizes the content into Lines. By default, each Line starts with a TitleCard, i.e. an editable video frame composed of text and images that explains the scenario illustrated with the subsequent shots. While TitleCards provide the context along the vertical dimension, the horizontal dimension is composed of video clips. After each TitleCard, the designer can add the corresponding clips following the description of the TitleCard.

The TitleCard has a dual function: it guides designers during the capturing process and it explains the scenario to the spectator of the final video prototype. Designers can edit TitleCards by tapping on them. VIDEOCLIPPER provides a simple TitleCard editor that allows direct manipulation of visual elements, such as text labels and images. These elements can be moved and resized with the provided handles (Figure 4.6a). Designers can also change the font size and the alignment of the text. Creating a TitleCard for each scene in the paper-based storyboard lets designers quickly transition from paper to VIDEOCLIPPER. VIDEOCLIPPER can import images taken with the tablet's camera directly into the TitleCard to reuse drawings and annotations from the paper prototype (Figure 4.6a). VIDEOCLIPPER automatically transforms each static TitleCard into a video clip, in the same way as title cards were used in old silent films. Each TitleCard has a default duration of two seconds that can be modified.



(a) Text can be align, resized and moved.

(b) Hand-drawn sketches can be imported.

Figure 4.6: VIDEOCLIPPER TitleCard's editor. The duration of the TitleCard can be edited with the button on the lower right corner.

To start recording, designers use the capturing view (Figure 4.7). During capturing VIDEOCLIPPER shows the storyboard information in a compressed layout surrounding the camera preview. On the left, only the initial TitleCards is shown in a single column. At the top, the linear sequence of clips from the currently selected Line are shown with an insertion marker, depicted by a blue vertical line. Designers can change the selected Line by tapping on the corresponding TitleCard

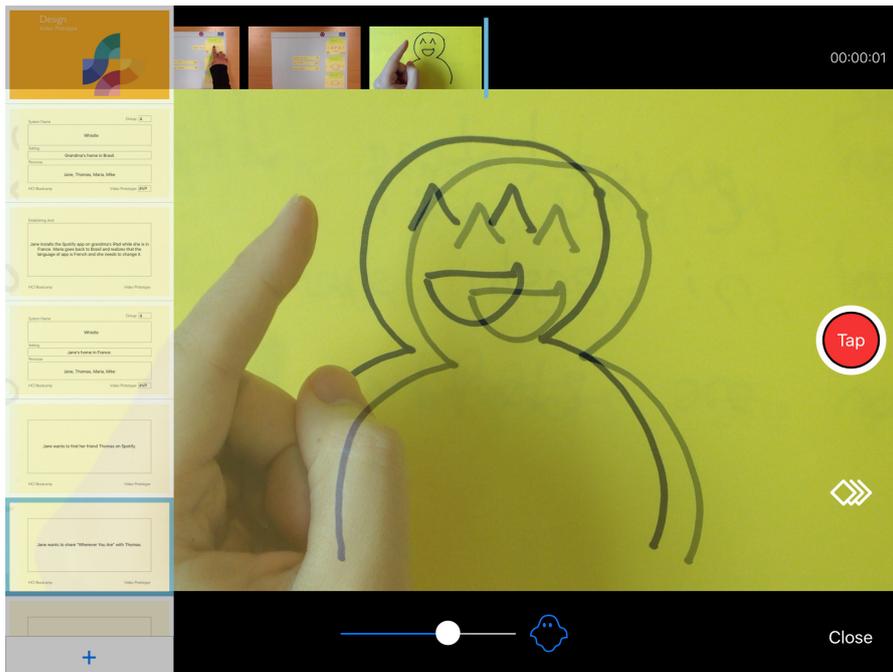


Figure 4.7: VIDEOCLIPPER's capture view. On the left, a column of TitleCards. On the top, the video clips of the selected Line. In the center, a preview of the camera, in this case with an active ghost image.

thumbnail on the left. At the center of the screen, VIDEOCLIPPER displays a preview of what the camera is currently capturing.

VIDEOCLIPPER helps designers create simple and rough "invisible cuts" without video editing. If an object should appear after the user presses a button, designers first record a clip showing the button press, then add the new object in the scene, and shoot a second clip with the hand starting in the same position. We call this technique "rough stop-motion" when it involves more than two clips. In both cases, VIDEOCLIPPER can display a "ghost image" (Figure 4.8) of the last frame of the clip before the insertion marker. This ghost image helps designers align the new clip by positioning every object—including the user's hands—in the same place as they were at the end of the first clip. The opacity of the ghost image can be controlled with a slider at the bottom of the screen. If the designers want to deactivate the ghost image, tapping the ghost icon toggles it. The ghost image, therefore, helps designers align two shots live without requiring them to do any post-editing.

In the storyboard view, video clips and Lines can be moved to reflect changes in the prototype. For example, to move a clip from one Line to another, designers can drag the desired clip directly. Also, if a Line should take place earlier in the final movie, designers can drag a complete Line (Figure 4.9), i.e. with all the associated video clips, to the new desired position. These actions can be seen as video editing, however, this approach is different than working



Figure 4.8: Adding a 'ghost' of the last recorded frame lets designers align the shots to create rough "invisible cuts".

with a paper storyboard and a video editing software. Unlike paper sketches, VIDEOCLIPPER automatically reflects changes made on the interactive storyboard directly into the final movie, which can be played at any time. Unlike video editing, VIDEOCLIPPER lets designers make semantic changes, such as moving a complete scene, instead of repositioning individual clips. We think this approach encourages designers to think in terms of a narrative instead of the technical aspects of video editing.

There are several examples of reusable artifacts in VIDEOCLIPPER. The ghost image reuses the last recorded frame as an alignment guide for a newly recorded clip. Designers can use this translucent view to create rough "invisible cuts" that depict the interaction, e.g., a tap of a finger that brings up a new window. One complete Line is also reusable to create ad-hoc alternatives in the prototype. In VIDEOCLIPPER, Lines can be duplicated to create variations of the scenario or the user's interactions. In such a branched story, one parent Line sets the scene while subsequent Lines depict alternative designs. Complete storyboards can also be cloned. This is useful to save alternative or intermediate versions or to compare prototypes after a re-design session.

At the end of the video prototyping session, VIDEOCLIPPER facilitates exporting the complete movie. When designers press play, all the TitleCards and clips are saved into an in-memory video file. Playing this video file starts from the timestamp corresponding to the selected Line in the storyboard. This lets designers evaluate the current context and move backward or forward in time to see the previous or next scenes. Like any other media file on the device, the created movie can be shared through any available file-transfer or streaming service.

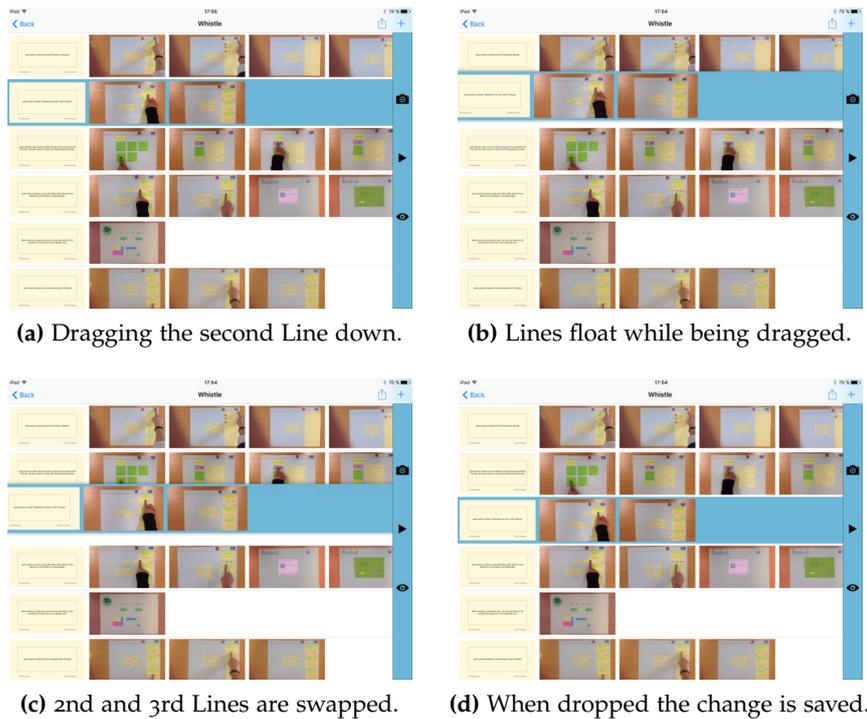


Figure 4.9: Moving Lines in VIDEOCLIPPER.

In summary, VIDEOCLIPPER embodies an integrated iterative design method that rewards discipline but permits flexibility for video prototyping. The tool provides a storyboard-style overview for each project, with TitleCards and thumbnail images, video capture for steady-state and rough stop-motion filming, editable, reusable TitleCards, and the ability to display different paths through the video or recombine videos in new ways for redesign.

4.3.1 Implementation

VIDEOCLIPPER is an iOS application designed for an iPad tablet. VIDEOCLIPPER uses Core Data to persist the data model (Figure 4.10) in an SQLite database. VIDEOCLIPPER relies on the AVFoundation framework (Apple Inc., 2018a) to manage time-based audiovisual media. Key uses of the framework include: transforming still images –screenshots of the TitleCards– into video tracks, composing video and audio tracks in memory to create a preview of the resulting video, and saving video files into the default Photo Album outside the application.

VIDEOCLIPPER includes four main screens (Figure 4.1): a list of Projects, a Storyboard view, a Capture view, and an Edit view –for TitleCards and video clips–. The Projects view lists the current projects, which can be played or opened into the Storyboard view. The Storyboard view displays a list of Lines, each with a TitleCard (on the left) and any number of video clips. Designers can prepare TitleCards in

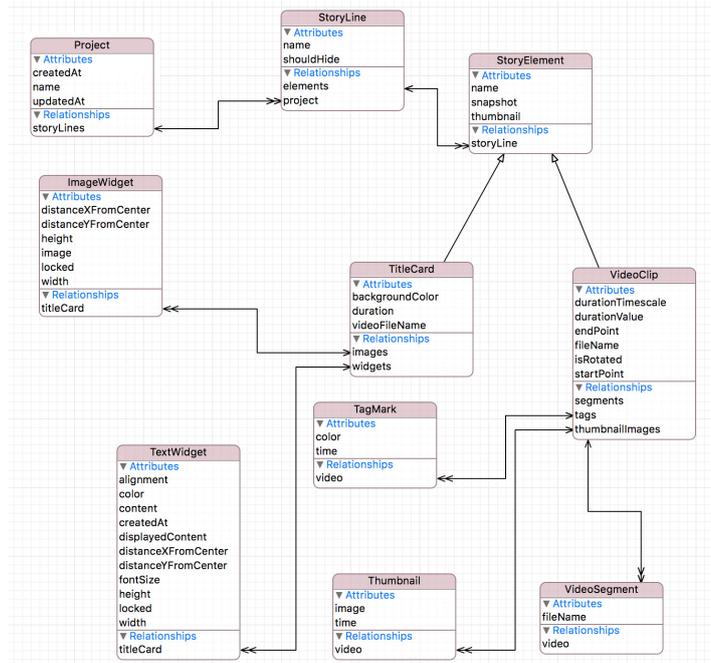


Figure 4.10: The VideoClipper data model.

advance to guide what to shoot next; new video clips appear at the end of the current Line. Lines may be added, cloned, rearranged, or deleted, and video clips may be dragged from one Line to another, as needed. Inspecting a video element opens a video player, with a non-destructive trimming tool, a trash button for deleting videos and four colored tag buttons to bookmark important frames. The vertical bar on the right has three buttons: The camera icon opens the Capture screen, the play button plays the current movie and the eye icon hides or shows the selected Line from the resulting video.

4.4 FIELD STUDIES

We deployed VIDEOCLIPPER during three intensive one-week design courses called Bootcamps, two 7-week design courses and several one-day interaction design workshops. I briefly summarize the post-analysis of three of these instances, highlight the most interesting findings and illustrate some of the video prototypes created by the participants.

4.4.1 First contact with real users

We first evaluated VIDEOCLIPPER in an intensive one-week Interaction Design Bootcamp with 34 students. This was the first time that VIDEOCLIPPER was put in the hands of real users. Also, this was the first introduction to video prototyping for these students. In this one-

week course, only two days are dedicated to video prototyping. As expected, users found many bugs, e.g., they dragged video elements to unexpected regions, crashing the application, and they captured video in multiple device orientations, breaking the final composition. We collected important feedback to improve the tool.

About four months after the Bootcamp, the same group of students participated in another design course that required the creation of a video prototype. On this occasion, we did not give them VIDEOCLIPPER, instead, we gave them a tablet device, i.e. an Apple's iPad, and let them pick the video tool of their choice. After their final presentation, we distributed a questionnaire to collect feedback (Appendix Section A.2). Our goal was to compare VIDEOCLIPPER with their tool of choice.

The students formed 8 groups of 4 or 5 members. Half of the groups captured video with the tablet device and its default camera application (4/8). The rest used their own DSLR cameras (3/8) or action cameras, such as the GoPro (1/8). The selected mobile video tools for the tablet groups were Apple iMovie (3/8) and Photodex ProShow Gold (1/8). The other groups imported their video files from their external cameras into a desktop application, such as Final Cut Pro (2/8), Adobe Premiere (1/8) and iMovie for desktop (1/8). Students chose video tools based on their previous experience, i.e. they preferred tools that they already knew.

We asked if they followed the paper storyboard that they created. Only one group followed the storyboard completely while the rest either "mostly followed the storyboard and only made a few changes" (5/8) or heavily modified the storyboard because "they changed their minds a lot" (2/8). One participant put it simply: *"While shooting we realized a few things were not clear"*. Another explained that the storyboard focuses on the user interactions but sometimes the viewer does not understand what is happening; for this reason, changes in the story are required to clarify what is the overall task that the user is trying to achieve. We asked this group how satisfied they were with the final result: *"Not that satisfied, because it's not easy to fake a precise interaction. We want to show the feedback of every action but it takes too much time."* Only half of the groups were satisfied with their final result.

Finally, we asked the students to reflect on their previous experience during video prototyping with VIDEOCLIPPER. We asked "Would VideoClipper have helped you? If so, how?". Most of the groups (6/8) replied that they missed the "ghost image": *"The main benefit [of VIDEOCLIPPER] is the 'transition' functionality [ghost image], so you can easily shoot stop-motion, without 'shaking too much'. Also, it makes the process much faster. In Premiere Pro you get lost in all the possibilities it offers and spend too much (most of the time) on unimportant details"*. Other students expressed the problem of gathering and editing video files instead of

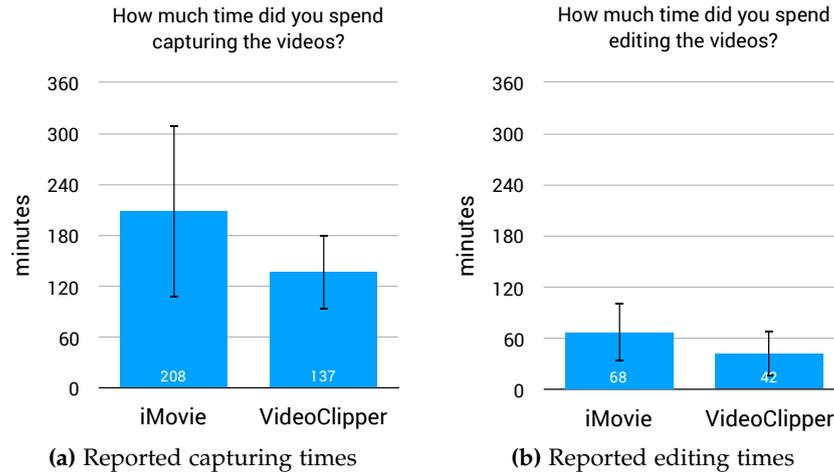


Figure 4.11: Reported average times in minutes collected from 16 students, the error bars represent the standard deviation. The average capturing time was 3h28m for iMovie and 2h17m for VIDEOCLIPPER. The average editing time was 1h8m for iMovie and 42m for VIDEOCLIPPER.

capturing on-the-go: “[With VIDEOCLIPPER there is] no need to gather the videos and then edit”.

By contrast, other people mentioned that editing is easier on a laptop computer: “Editing on [a] laptop is easier than on iPad. So it will be nice if we can do the cuts and edits on a computer.” We agree with this problem but not with the solution, since the goal of rapid video prototyping is to reduce the editing instead of encouraging it. Some students preferred editing with desktop tools because they provide a higher level of visual fidelity: “The fading picture [ghost image] is useful but without VIDEOCLIPPER you have nicer TitleCards, it’s easier to shoot in a different order if necessary.” However, video prototyping should focus on the user’s interactions rather than on the quality of the visuals. This reinforces the idea that VIDEOCLIPPER is a tool that supports the values of rapid prototyping rather than professional video editing. VIDEOCLIPPER should not be seen as a standalone video editing tool but as a video tool that supports a rapid video prototyping process.

4.4.2 Comparing mobile iMovie with VideoClipper

In a 7-week course on Designing Interactive System, 20 students prototyped two interactive systems, an captured both designs in a video prototype. We provided the students with a tablet and the mobile version of iMovie as a tool for their first video prototype. For their second video prototype, we asked the students to use VIDEOCLIPPER. After they finalized the course we distributed a questionnaire (Appendix Section A.2).

The students formed 6 groups of 3 or 4 members. One of the groups did not complete the questionnaire so we are only reporting the results from the remaining 16 students. We asked participants how much time they spent during capturing, i.e. recording video, and editing, i.e. creating effects, trimming and moving clips (Figure 4.11).

Students reported that they spent more time capturing video with iMovie ($M = 208.13$ minutes, $SD = 100.68$) than with VIDEOCLIPPER ($M = 136.89$ minutes, $SD = 43.13$). They also reported that they spent more time editing video with iMovie ($M = 67.50$ minutes, $SD = 34.21$) than with VIDEOCLIPPER ($M = 42$ minutes, $SD = 26.14$). As expected, participants tended to spend more time capturing and editing with iMovie.

When we asked them about the biggest problem they encountered with iMovie, participants complained about collecting the meaningful clips from all the unordered recordings:

"The most difficult part is actually before the shooting. Writing scripts and drawing storyboards that made sense and explain the design perfectly. But there are lots of wasted clips [...] It was really annoying to pick out valid ones from so many clips."

"Sometimes we need to shoot a scene more than 1 time. In that case, it could take a while for us to find the right video clip for editing."

They also mentioned some other important problems with iMovie during editing:

"When I had to move the clips from one side to any other side. There is only one queue and it is super time-consuming to hold one piece of clip and wait until it goes to the other end then realize it."

"I cannot group the pieces. It leads to that I have to change the order one by one, although the pieces should stay together."

"Rework of paper prototype to keep consistency."

"Realizing something on your interfaces was misplaced."

We also asked about their biggest difficulties with VIDEOCLIPPER. Besides some unexpected crashes, participants complained about finding clips in the interactive storyboard:

"It's not easy to get the general idea of every clip just by the [thumbnail] picture from the clip. Because the neighboring clips look really similar. And I have to play the clip to know which one it is."

Some participants missed editing features, such as cutting a clip:

"Having forgotten to add a clip between an action, no split possible".

Students generally reported that *"the storyboard overview is really helpful"* and, similarly to the first study, that the "ghost image" was highly appreciated:

"We are extremely satisfied by the final result. We especially appreciate the 'ghost feature' it was so efficient and helped us saving time an make good quality of transitions."

While editing activities were present in both VIDEOCLIPPER and iMovie, it is important to observe that students considered any change in the final video as editing. For example, editing with iMovie includes moving clips but also trimming, cutting or adding effects. VIDEOCLIPPER features minimal video editing capabilities such as non-destructive trimming and moving clips. However, we consider moving Lines in the storyboard to be a narrative form of editing, much different from moving individual clips in iMovie. We asked students what type of editing they performed in VIDEOCLIPPER: 30% used trimming, 35% moved TitleCards, 40% moved Lines, and 60% moved clips. Re-organization of video clips is still the most common edit, both in iMovie and VIDEOCLIPPER, however, moving Lines was extremely appreciated and was ranked as the second most used type of edit.

4.4.3 Using VideoClipper in a longer course

Finally, we assessed VIDEOCLIPPER in the context of a 7-week design course. The students formed 12 groups, one group of 2 and 5 members while the rest had 3 or 4 members. In this case, the allocated time for video prototyping was one week, while the previous intensive courses only dedicated two days to video prototyping. Similarly to the previous examples, after their final presentation, we distributed a questionnaire to collect feedback ([Section A.2](#)).

We observed similar trends in capture and editing time even though students had more total time for finishing the video prototyping activity. In terms of capturing time, 65% of the students spent less than 3 hours recording but 35% spent between 3 hours and 9 hours ([Figure 4.12a](#)). In terms of editing time, 88% of the students spent less than 3 hours modifying the video and all of them (100%) spent less than 4 hours 30 minutes ([Figure 4.12b](#)). It seems that regardless of the extra time, designers did not spend more time editing in the long course than in the short course.

We detected recurring patterns while creating video prototypes with VIDEOCLIPPER. First, participants tried to avoid re-shooting by

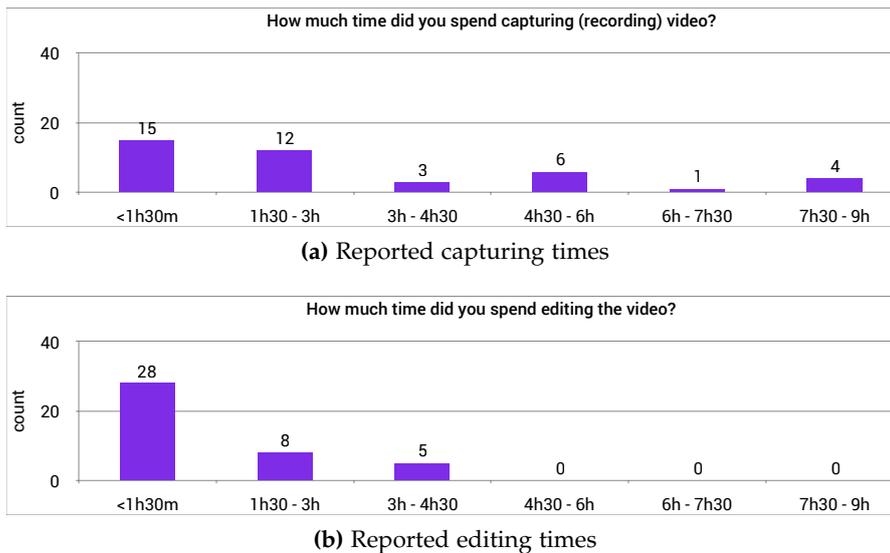
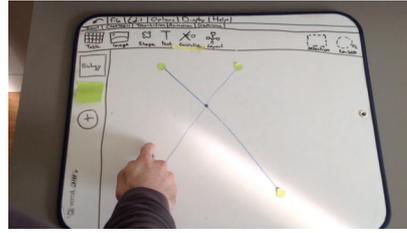


Figure 4.12: Reported times collected from 41 students using VIDEOCLIPPER in a longer course. The average capturing time was 3 hours and the average editing time was 47 minutes.

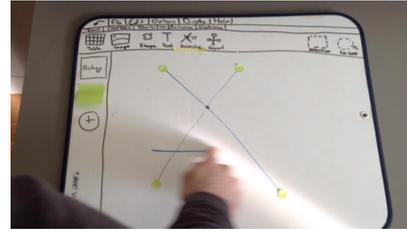
re-purposing continuous physical actions as the system visual outputs. For example, a group hid a pen under the actor's sleeve to illustrate the inking feedback of a touch-based drawing interface (Figure 4.13). In this way, designers can represent the inking feedback with only one shot without the need to use rough stop-motion. To illustrate the detection of vertices, they added an invisible cut at the end of the scene (Figure 4.13d).

Participants also created reusable backgrounds to represent the static aspects of the interface. This can be seen in the previous example, where the whiteboard includes static interface elements, such as the menu bar and a side toolbar (Figure 4.13a). Then, changes to the interface are only made in the center of the whiteboard. Another group prototyped an interaction with a draggable and resizable lens (Figure 4.14). Designers created a map as the background, where the paper prototype of the lens can be freely moved (Figure 4.14a). However, they wanted to reduce the size of the lens with a pinch gesture (Figure 4.14b). To create this effect, designers created a new paper prototype of a smaller lens and introduced an invisible cut after performing the pinch gesture (Figure 4.14c).

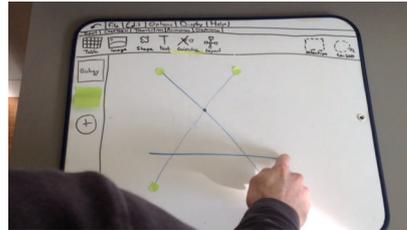
Finally, designers created mix-fidelity representations according to the aspects of the prototype that they wanted to highlight. One group created a paper prototype of the background needed for their desktop application. Afterwards, they displayed a picture of this paper prototype on a laptop to create a background for their video prototype (Figure 4.15a). Another group was prototyping a video application. Showcasing dynamic video with paper sketches was difficult, so they used a smartphone screen as a way to illustrate video playback. They



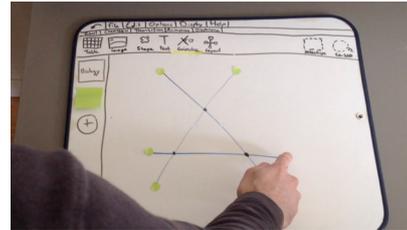
(a) The actor hides a pen on his sleeve to draw with continuous feedback.



(b) The actor simulates drawing a straight line with the index finger.



(c) The actor maintains the final position.



(d) An invisible cut adds the vertex recognition feedback.

Figure 4.13: A video prototype of a touch-based drawing application with vertex recognition.

cut a rectangle in their paper prototype and positioned a smartphone with a video player right underneath this opening (Figure 4.15b).

Following the same idea of mix-fidelity prototyping, another group was prototyping an interaction with a plant (Figure 4.16). They mixed in the same prototype a real plant (Figure 4.16a) and a paper prototype of the connected system (Figure 4.16a). When the user touches a plant's branch the system retrieves information associated with the time when the branch had the size indicated by the user's touch position.

Finally, all the groups reused the video prototypes for two activities: re-design and presentation. For re-design, students shared their video prototypes, within and across groups, in order to get feedback, discuss design directions or ask for clarifications. For presentation, at the end of the course, each group presented a design brief and did a voice over during the projection of their video prototype (Figure 4.17). This enabled all the students in the course to see the results of the design session and enabled everyone to have concrete and rich discussions.

4.5 LIMITATIONS AND FUTURE WORK

In terms of technical limitations, VIDEOCLIPPER is only available for the iOS platform. This restricts the participation of impromptu designers with other devices and platforms. An interesting direction for the future is to evaluate how to support web-based video technologies.

VIDEOCLIPPER was conceived as a lightweight tool, i.e. it supports a limited amount of professional video editing workflows. However, I

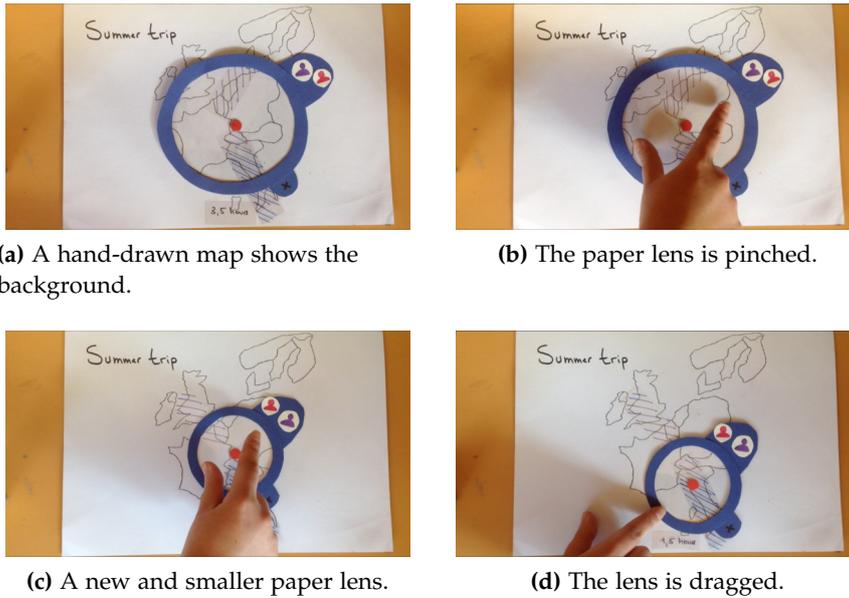


Figure 4.14: A video prototype of a draggable and resizable lens.

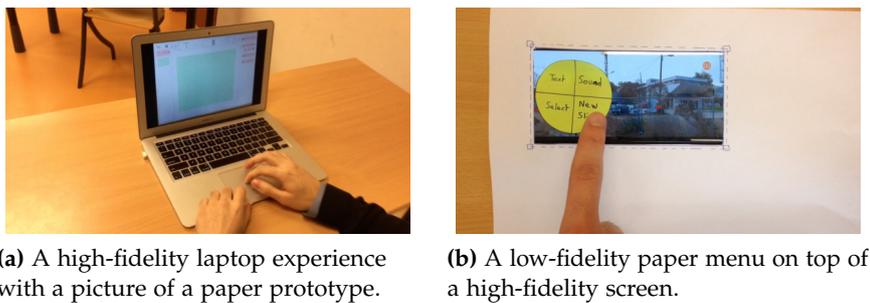


Figure 4.15: Students mixed different fidelities to illustrate multiple aspects of the design.

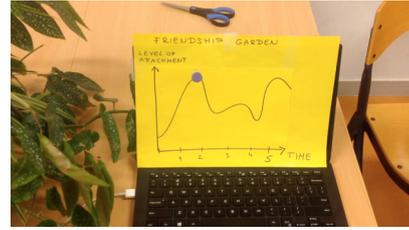
see this limitation as a strength: VIDEOCLIPPER is more aligned with a rapid video prototyping process than with a video editing activity.

VIDEOCLIPPER was generally manipulated by a group of designers but the tool itself is restricted to a single user. This limits the parallelization of tasks and the simultaneous exploration of multiple ideas. While video files can be recorded with other cameras and imported into VIDEOCLIPPER, supporting multiple users could provide extra benefits, such as the inclusion of remote designers and the synchronization with multiple devices.

Even though we saw instances of reuse, the tool only supports a few reusable objects. VIDEOCLIPPER supports ephemeral reuse, e.g., with the ghost image, and reuse by copy, e.g., by letting users clone Lines. However, this type of reuse does not support the automatic propagation of changes in the original object to the copies. We could use references between Lines to indicate an inheritance relationship,



(a) The user-actor interacts with an actual plant.



(b) The system outputs are represented with a paper prototype.

Figure 4.16: A video prototype of an interaction with a real plant.

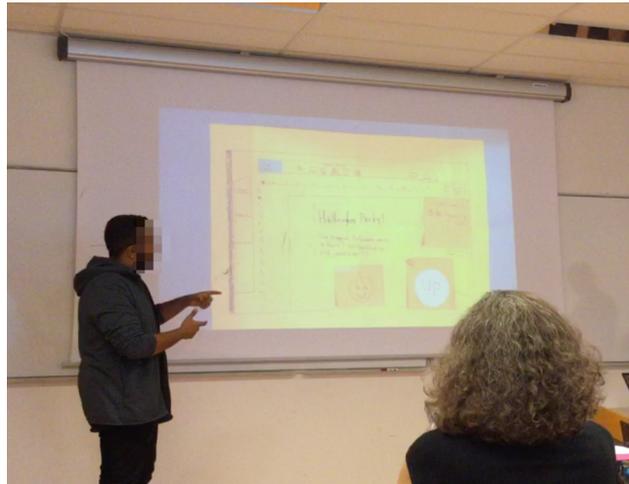


Figure 4.17: A student presenting on top of the video prototype playing in the background.

e.g., when a Line illustrates the context and subsequent Lines illustrate alternatives. Reuse by reference could simplify the propagation of changes from the context to all the alternatives. However, it would also introduce extra complexity during rapid prototyping.

Finally, users exhibited recurrent patterns to reduce the amount of re-shooting needed. For example, by generalizing the common background and only changing the interface or by mixing different levels of fidelity. I think that digital tools could be combined with low-tech prototyping to minimize the need for re-shooting and increase reuse.

MONTAGE: COMBINING VIDEO PROTOTYPES TO PROVIDE REUSE

VIDEOCLIPPER focused on the capturing phase and provides a simple model for compiling the final video prototype, i.e. just straight cutting of the recorded video clips. I identified two main barriers to using video in interaction design: time to capture and time to edit (Section 3.2). VIDEOCLIPPER supports planning as an interactive storyboard to streamline the capturing process and reduce the need for editing.

However, VIDEOCLIPPER's users constantly asked for more facilities to reduce unnecessary work. They created layered ad-hoc approaches to reuse artifacts. For example, they established a background that was repeatedly reused with different user actions changing on top of it. Another technique was mixing paper representations with other props, just as mobile screens and laptops devices. Also, in the questionnaires participants asked for more digital support to facilitate changes over their design.

These user behaviors inspired me to explore a new type of tool, one that already supported this layered approach to decouple background, i.e. context of use, and user interface.

5.1 DECOUPLING CONTEXT AND INTERFACE WITH MONTAGE

MONTAGE is composed of a central device –the *Canvas*– connected to two mobile devices –*UserCam* and *WizardCam*– with video streaming and recording capabilities (Figure 5.1). These devices, typically phones or tablets, are used as remote cameras. They stream, either in parallel or independently, the *context* of use where the user could interact with the prototype and the prototyped *user interface* itself. The *Canvas* lets designers organize and compose the video segments, and augment them with digital drawings that can be re-shaped and modified. Interaction designers can compose, draw and modify the prototype during rehearsal, during recording, or after filming. MONTAGE focuses on low-budget video recording but provides designers with features currently only available in high-budget video prototyping, such as layering and tracking. Interaction designers can start with traditional paper prototyping and progressively move towards modifiable and re-usable digital representations without the need for professional video editing software.

MONTAGE targets interactions that require continuous feedback, such as scaling a picture with a pinch or selecting objects with a lasso,



Figure 5.1: An overview of the MONTAGE video prototyping system. Here, two designers, a wizard and a user-actor, collaborate side-by-side. The *UserCam* captures the user-actor enacting user inputs in the actual context, i.e. using his phone at the office. The *WizardCam* captures the paper prototype and the *Canvas* captures the wizard's digital sketches.

which are often challenging to perform with traditional paper and video prototyping. I first illustrate the approaches and challenges of prototyping continuous feedback with traditional video prototyping, and then present an enhanced approach using Mirror, a mode of MONTAGE that mixes streamed physical elements (such as a paper prototype) captured by a camera, with digital elements created remotely by a wizard. Finally, I present MONTAGE Chroma to reduce re-shooting while exploring alternative designs.

5.2 LIMITATIONS WHEN COMBINING PAPER AND VIDEO

Imagine a group of designers prototyping an interaction technique with dynamic guides, similar to OctoPocus (Bau and Mackay, 2008). OctoPocus provides continuous feedback (inking) and feedforward (potential options) of the gestures as a user performs them¹. The designers want to illustrate the use of this technique in the office, when interacting with the profile picture of a friend on a phone: when dwelling on the picture, OctoPocus should show three gestures for calling, messaging or finding directions to the friend. The designers print an image to use as the profile picture and attach it to the screen of a phone, used as a theatrical prop, to contextualize the prototyped interaction. The user-actor draws on the profile picture to mimic the continuous feedback of a gesture. She uses a black pen hidden as well as possible in his palm, while a wizard draws the feedforward, i.e. three colored curved lines.

¹ See <https://vimeo.com/2116172>

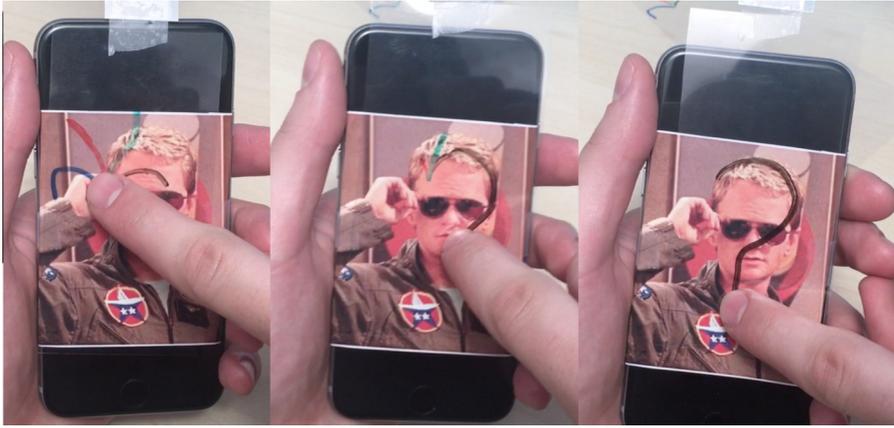


Figure 5.2: OctoPocus with traditional video prototyping. The designers create a rough stop-motion movie with only four stages of the interface, resulting in a poor representation of the dynamic interaction.

This approach to prototyping continuous feedback and feedforward has three main drawbacks:

- The hand and pen of the wizard appear in the video;
- The profile picture has drawings on it that might not be easy to erase, so in case of mistakes or changes, it requires a new picture or at least re-recording the whole video; and
- Illustrating the use of the same technique in different contexts (on the profile picture of other friends, or in a completely different scenario) also requires re-shooting.

The designers take a different approach to avoid these problems. They create four sketches with transparent paper to represent the different stages of the OctoPocus interaction (Figure 5.2): They plan to reveal the feedforward and feedback progressively by overlaying the sketches on top of the profile picture, one sketch at a time. They use a mobile device on a tripod to record a rough stop-motion video of the interaction.

With this approach, the designers reduce the presence of the wizard in the video, as they place the sketches on top of the profile picture in-between the stop-motion takes. Because the sketches are drawn over transparent paper instead of the profile picture, the designers can reuse their prototype props to illustrate other contexts of use. Nevertheless, this approach also comes with limitations and drawbacks:

- While it is possible to reuse the sketches in other contexts, the whole interaction needs to be re-shot;
- A sequence of four sketches will poorly communicate the highly continuous nature of the interaction; and

- Making a stop-motion video shifts the designers' attention from experiencing and reflecting on their design to coordinating sequences of extremely brief video shots.

A third approach is to use video editing software instead of paper sketches to add a digital overlay with the user interface on top of a previously recorded shot of the user actions. This approach has the disadvantage of creating a disruptive context switch, from a design session that does not require specialized skills to a video editing session requiring trained editors. Also, with paper prototyping the user interface is partially hidden by the user's hands, so a simple digital overlay will not produce the same effect. An experienced video editor could simulate the fact that the interface is below the user's fingers by creating a mask of the user's hands at several keyframes, e.g., by rotoscoping or using specialized software.

In summary, with current approaches to video prototyping, designers struggle to represent continuous feedback and to reuse prototype props and previously captured interactions. The tools that address these problems require video editing skills and extra effort in post-production, interrupting the flow of the design process. I want to better support video prototyping without disrupting the design process nor requiring specialized video editing skills.

5.3 MONTAGE MIRROR: PROTOTYPING CONTINUOUS FEEDBACK

MONTAGE Mirror mixes physical elements captured by a *WizardCam* (Figure 5.3a and Figure 5.3b), with digital sketches drawn remotely in the *Canvas* (Figure 5.3d). The user-actor's phone displays a video stream of the paper prototype combined with the digital sketches—the *interface*. As the user-actor interacts with the phone, the wizards provide live feedback by editing the digital sketches on the *Canvas* or by manipulating the paper prototype captured by the *WizardCam*.

For example, to prototype OctoPocus, the user-actor sees the profile picture on his phone, captured by the *WizardCam*. As she performs a gesture on the screen, she sees the feedback and feedforward sketched remotely by the wizard on the *Canvas*. In this way, the user-actor can experience the prototyped interaction without the hands of the wizard getting in the way. The *UserCam* captures the interaction over the *interface* and the *context* of use to create the final video prototype.

Designers can animate changes in position, size, rotation angle, color, and thickness of the digital sketches without the tedious coordination required by stop-motion videos. Digital sketches can be grouped to create compound objects that have semantic meaning in the story. Moreover, thanks to the *WizardCam* stream, traditional paper prototyping techniques are still available if necessary: physical sketches and props added to the paper prototype are directly streamed to the user-actor's phone. For example, after the user-actor performs

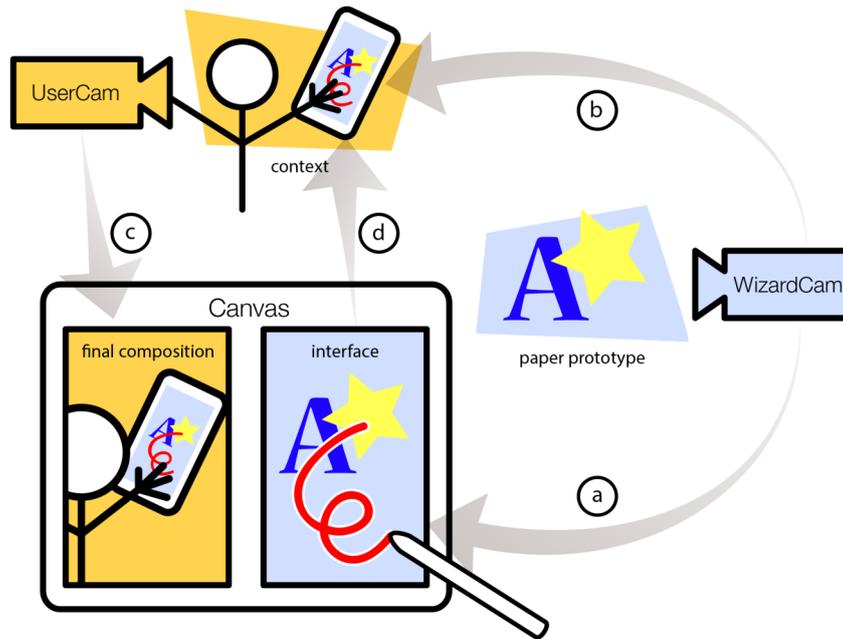


Figure 5.3: Mirror mode: the *WizardCam* live streams to both, the *Canvas* (a) and to the prototyped device (b) in the *context*. The *UserCam* only streams to the *Canvas* (c). Finally, the *Canvas* sends the sketches to the prototyped device to complete the mirroring of the *interface*

a gesture with *OctoPocus*, the wizard can add a sticky note with the text “Calling Barney” on top of the profile picture.

MONTAGE Mirror augments video prototyping with *live* digital sketches. In the *Canvas*, designers use the stylus to draw sketches and perform simple actions, such as pressing a button. Designers can move, resize and rotate sketches with the standard pan, pinch and rotate gestures. Unlike stop-motion videos, digital sketches allow prototyping continuous feedback interactions that look fluid and allows designers to focus on the design process instead of coordinating complex wizard actions. For example, to prototype the drawing of a question mark and the inking feedback, the wizard draws at the same time that the user-actor is gesturing. The designer rewinds the recorded video to the point where she wants the dynamic guides to appear and draws them. After pressing play, she uses a slider of the sketch interface to make the stroke progressively disappear as the video plays (Figure 5.5).

Mirror mode supports the prototyping of dynamic interfaces and continuous feedback. Nevertheless, it still requires re-shooting when exploring alternative designs or contexts, e.g., showing *OctoPocus* on something else than a mobile phone. Designers have to record the whole video again even for small changes, such as changing the color of the dynamic guides.

5.4 MONTAGE CHROMA: REUSING CAPTURED INTERACTIONS

To help designers reuse previously captured interactions and reduce re-shooting, the Chroma mode takes advantage of a well-known video editing technique called *chroma key compositing*. With *chroma keying*, the subject is recorded in front of a solid background color, generally green or blue, and this background is replaced in post-production with the desired content. This technique is commonly used by weather presenters on television, to replace a green background with an animated map with weather information. In our example, the user drawing a question mark is recorded over a phone showing a green screen, which is later replaced with the prototype interface. Achieving a clean chroma keying requires special attention to proper lighting conditions and using the right shade of green. However, I am not concerned with achieving a perfect result during an early-stage low-fidelity prototype. I can also use a different color than green, as long as it is distinct enough from the rest of the scene, by selecting it in the video feed with MONTAGE's color picker.

In order to replace only the portion of the screen that contains the *interface*, I display a green screen on the user-actor's phone. The *UserCam* records the final video prototype, but in Chroma mode MONTAGE also tracks the four corners of the green screen and sends this data to the *Canvas*. Then, the *Canvas* performs a perspective transformation of the current frame of the *interface* and replaces the green area with the transformed *interface*. MONTAGE Chroma not only performs this composition in post-production, i.e. after recording, but also during recording, in the *final composition* live preview.

Designers simply need to rewind the recorded video to add new sketches or modify existing ones. They can draw or modify the sketches over the *interface* or over the *context* to annotate the story, e.g., with user reactions or speech bubbles. Designers do not need to re-shoot the *context* to make small changes to the *interface*, or vice versa. In the OctoPocus example, after recording a version of the prototype, designers can add new dynamic guides without re-shooting by simply drawing over the recorded video. The new changes are immediately available in the *Canvas final composition* live preview.

The setup of the system in Chroma mode (Figure 5.4) has just one difference from the setup in Mirror mode. The *UserCam* still captures the *context* but the phone now shows a green screen (Figure 5.4a). The *final composition* preview in the *Canvas* shows the chroma keyed result, positioning the *interface* correctly under the user's hands and following the phone's boundaries. When the designers modify the paper prototype or the digital sketches, i.e. add, remove, move, rotate, scale, or color change the sketches, MONTAGE Chroma shows these modifications immediately in the *final composition* live preview, but not on the actual phone.

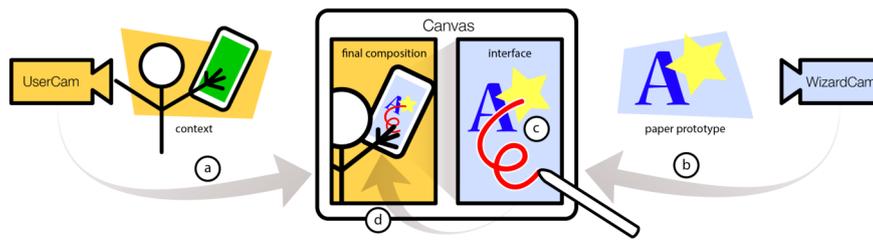


Figure 5.4: Chroma mode: the *UserCam* captures the *context* (a) and the *WizardCam* captures the *paper prototype* (b); Both live-stream video to the *Canvas*. Designers draw digital sketches over the streamed *paper prototype* to represent the *interface* (c). In the *Canvas*, the green screen is replaced with a perspective transformation of the *interface* to create the *final composition* (d).

When possible, I recommend using MONTAGE Mirror during rehearsal and MONTAGE Chroma during recording. If the *interface* is to be placed on a screen-based device with streaming capabilities, using MONTAGE Mirror lets the user-actor experience the prototype directly. During recording, using MONTAGE Chroma allows to create composable video segments and lets the *interface* and the *context* to be changed independently. MONTAGE Chroma provides a “user overlay” to assist wizards sketch in relation with user inputs. For example, when the user-actor places a finger over the green screen, the wizard can see a translucent image of this finger over the drawing area of the *Canvas*; this helps wizards better coordinate spatially and temporally the drawings with respect to the user inputs. MONTAGE uses the inverse perspective transformation of the green screen to correct the placement of the overlay.

5.4.1 Exploring design alternatives and multiple contexts of use

With chroma keying, the recorded *interface* videos and *context* videos can be changed independently. This flexibility reduces the cost of exploring different design alternatives and multiple contexts of use. For example, once an *interface* video of the OctoPocus technique is prototyped, it can be embedded in multiple *contexts* (Figure 5.6): different videos can show the user-actor using OctoPocus in the metro, in a park or at a party without having to re-shoot the interaction technique. The other way around is also possible: Several alternative designs of the *interface* can be embedded in the same *context*, with different videos showing the original context of the actor-user in his office using OctoPocus on a social media profile, an email client or the camera app.

Besides recording with the *UserCam* in different places, i.e. in a park or an office, the *context* can be changed by using a different device. MONTAGE Chroma works with any device that can display a solid color in a rectangular frame, such as watches, phones, tablets, laptops

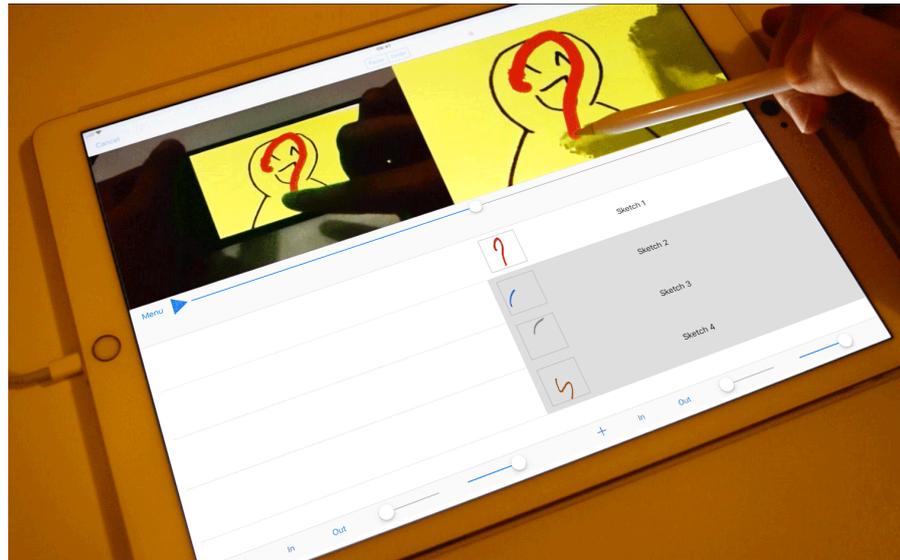


Figure 5.5: The *Canvas* sketching interface: The *final composition* (left) and the *interface* (right) show the “user overlay”. Both sides have a list of sketches and animation controls at the bottom. The in/out buttons make the selected sketches appear/disappear. The sliders control the stroke-start, now at 0%, and the stroke-end, now at 100%.

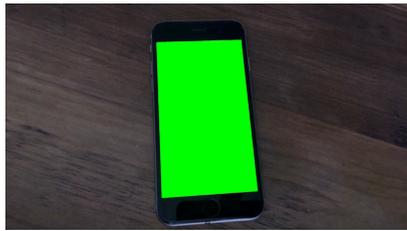
and even wall-size displays, enabling designers to explore multiple display alternatives.

5.4.2 Supporting multiple interaction styles

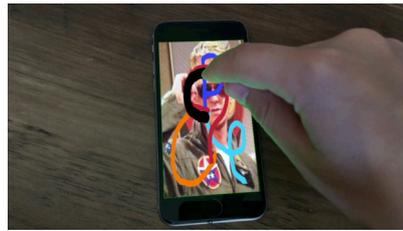
MONTAGE Chroma is not limited to displays such as a phone’s screen. Designers can video prototype over other rectangular surfaces, such as boxes, books and whiteboards with a solid color. MONTAGE can prototype gesture-based interactions over a green sticky note or a t-shirt stamp. This allows the exploration of stationary as well as mobile contexts, e.g., sitting in front of an interactive table or walking with a phone.

MONTAGE’s digital sketches can depict 2D widgets common in WIMP interaction such as buttons, sliders and menus. Toggling the visibility of sketches on or off (Figure 5.5) at precise moments in the video is ideal for prototyping interactive transitions of the interface states, e.g. idle/hover/press states of a button or the screen-flow of a mobile app. Static sketches can depict discrete feedback, e.g. adding an object after pressing a button, while animated sketches can depict continuous feedback, e.g. inking, dragging or resizing with a pinch.

MONTAGE also supports prototyping interactions that involve indirect input devices. By using a green screen on a laptop’s display designers can still see the mouse cursor, facilitating the positioning of interface elements. For example, to prototype a marking menu,



(a) A mobile phone with a green screen to illustrate a stationary context.



(b) The interface replaces the green area with the user inputs on top.



(c) A green screen over the phone is replaced with the same interface.



(d) The user-actor explores the same interface over a green screen in a watch.

Figure 5.6: Green screens let designers explore the same interface in multiple contexts, e.g., stationary on a desk or walking with a phone or a watch.

designers create a simple paper prototype of the menu. When the user clicks, designers pause the recording and introduce the paper prototype under the *WizardCam*. Designers resume recording and when the user moves, the wizard adds digital sketches to illustrate the feedback. Finally, when the user selects an item, the wizard highlights the corresponding item and modifies the paper prototype to show the response of the system.

With MONTAGE designers can even prototype interactions that use the spatial relationship between the devices and the users, such as Proxemic Interaction (Greenberg et al., 2011). For example, a user can walk closer or farther away from a device tracking her location, while the wizard manipulates the sketches to react to the user's position.

Prototyping multimodal interfaces, e.g., voice interaction and body movement, is possible with MONTAGE. For example, to re-create the foundational Put-that-there interaction (Bolt, 1980), both cameras record video and audio so that designers can enact the voice interactions that will be recorded. After the actor utters a voice command, the wizard pauses the recording, adds the necessary digital sketches and resumes recording. The video can then be modified, without re-shooting, to transition from paper to digital sketches or to add more details, such as the on-screen cursor. In more complex scenarios, the designers can sketch annotations on the *context* to indicate the user's voice commands or the system's audio responses, or to represent haptic feedback from the system, such as vibrations.

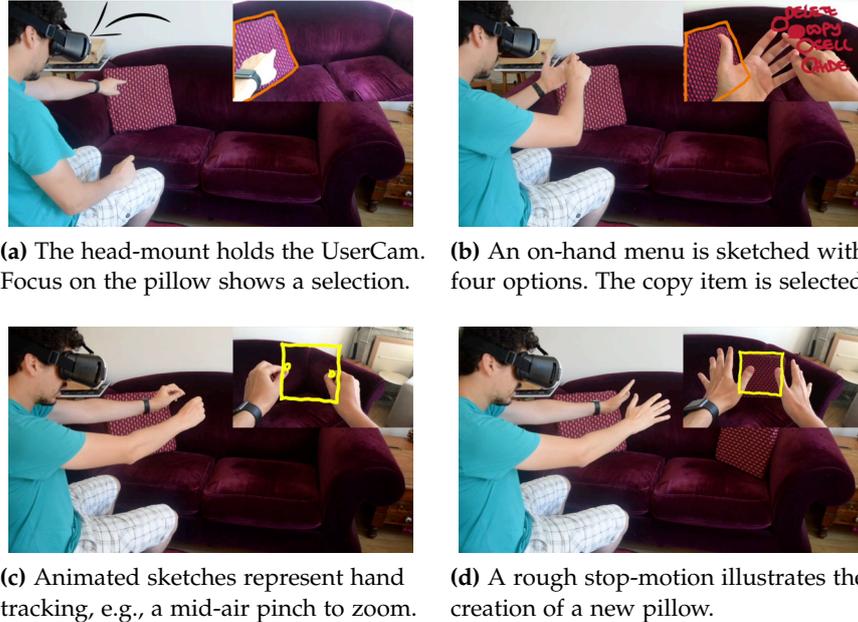


Figure 5.7: Sketches over the context can represent AR visual elements.

Video prototyping AR systems is also easy with MONTAGE (Figure 5.7). The designer attaches the *UserCam* to a headset, simulating the use of smart-glasses. With this setup, the user-actor has his hands free to interact with the overlaid images created by the wizard. In many cases, there is no need to use Chroma mode because the interface elements are overlaid over the camera feed. For example, to prototype a menu that follows the user's hand, the wizard only needs to sketch over the *context* and move the sketches to follow the hand's movements. The final video prototype will show the interaction from the point of view of the user-actor, i.e. the *UserCam*.

5.5 IMPLEMENTATION

MONTAGE currently runs on iOS version 11.2. In our preferred setup, the *Canvas* runs on an iPad Pro 12.9 inches (2nd generation) with an Apple Pencil stylus. Generally, the *UserCam* runs on an iPhone 6S and the *WizardCam* on an iPad Mini 3. Other wireless devices are also suitable as cameras and mirrors. I tested MONTAGE Mirror with an Apple Watch 1st gen. (42mm case) and a MacBook Pro (13-inch, 2015).

I use AVFoundation (Apple Inc., 2018a) to capture video, intercept frame buffers, and create movie files. The frame buffers are processed with Core Image (Apple Inc., 2018b) to clean the images before detecting rectangular areas, to perform perspective and correction transformations, and to execute the chroma keying.

I use a zero-configuration network where the *Canvas* acts as a server browsing for peers that automatically connect to the system. Each

camera records its own high-quality movie, currently 1280x720 pixels. However, the video stream is sent at a lower quality (480x360 pixels) to maintain an acceptable latency during rehearsal and recording ($M=180\text{ms}$, $SD=60\text{ms}$)². The devices' clocks are synchronized to start, pause, and end the recording at the same time. Due to delays introduced by the wireless connection, I created a protocol to let the devices synchronize: When the designer presses *Record* on the *Canvas* the screen displays a "3, 2, 1, go" animation. This delay lets devices prepare for recording and synchronize their capture start time. I use the same mechanism when the designer resumes the recording after pausing.

In order to create a movie combining the dynamic sketches with the captured video, MONTAGE saves the designers' inputs during the manipulation of the digital sketches. MONTAGE uses this information to create keyframe animations at different points of the video playback. I added a synchronization layer on top of both to link these animated sketches with the underlining movie player. This new layer coordinates the animation playback with the movie file playback.

5.6 LIMITATIONS AND FUTURE WORK

I have observed that Chroma mode works best with flat surfaces, and rectangle tracking works poorly with flexible or shape-changing surfaces. Also, excessive user occlusion can prevent proper screen tracking. As a workaround, when the device is not moving, designers can lock the last tracked rectangle. MONTAGE Chroma can also replace any solid-color area, regardless of its shape, with the *interface*. However, without position tracking, the perspective transformation of the interface is lost, resulting in a "naive" chroma keying.

Future work involves evaluating MONTAGE with professional interaction designers, improving screen tracking and exploring the reuse of video prototypes beyond early-stage design, e.g. by supporting the transition to high-fidelity prototypes.

One drawback of chroma keying is that the user-actor interacts with a green screen, not the final prototype. Using Mirror mode during rehearsal mitigates this problem. In Chroma mode, the user-actor should see the *Canvas* in order to monitor the state of the *interface* in relation to his inputs.

Finally, MONTAGE only supports interaction styles that can be mimicked with video. Currently, MONTAGE cannot illustrate complete immersive virtual worlds, e.g., VR applications or 3D games. However, MONTAGE can still video prototype particular aspects of these examples, such as hand tracking.

² for this benchmark, I used 20 samples taken randomly during a 30 minutes recording session

5.7 CONCLUSION

Current approaches to video prototyping make it difficult to represent continuous feedback and reuse previously captured interactions. The tools that address these problems require video editing skills and extra effort in post-production, interrupting the flow of the design process. Therefore we need to better support video prototyping without disrupting the design process nor requiring specialized video editing skills.

I presented MONTAGE, a distributed mobile system that supports video prototyping in the early stages of design. Our technical contribution is a novel tool integrating multiple live video streams, screen tracking, chroma keying, digital sketches and physical prototyping in one fully mobile video prototyping system. While these individual techniques are not new, MONTAGE combines them in a novel way to support an enhanced video prototyping process: rehearsal with Mirror mode and recording with Chroma mode. MONTAGE Mirror augments video prototypes with remote digital sketches, while MONTAGE Chroma supports the reuse of previously recorded videos to explore alternative interfaces and contexts of use. I described a scenario that demonstrates the limitations of traditional paper and video techniques, and showed how MONTAGE addresses them. Finally, I illustrated how MONTAGE can prototype a variety of interaction styles, including touch-based (OctoPocus), WIMP (marking menu), AR (on-hand menu), multimodal (Put-that-there), and ubiquitous computing (Proxemic Interaction).

Part II

INTERACTIVE PROTOTYPING

In collaboration with Nolwenn Maudet³.

³ Part of this work is in submission to ACM Transactions on Computer-Human Interaction (TOCHI). Both, Nolwenn Maudet and I, participated in the preparation, execution and analysis of all the studies, as well as the design of the ENACT tool. Nolwenn created all the StoryPortraits used during analysis. I was the principal developer of the ENACT tool.

WHY DESIGNER-DEVELOPER COLLABORATION?

Interactive software development in the 1970s and 1980s involved traditional software engineering processes, with limited interdisciplinary collaboration between developers and graphic designers. By the 1990s, the advent of full color, high-resolution displays enabled high-quality interactive graphics, with a corresponding need for professional designers. Today, graphic designers, interaction designers and user experience specialists are routinely part of the teams creating interactive software, together with software developers. However, integrating designers' and developers' work practices has proven difficult, often leading to friction between them (Ferreira, Sharp, and Robinson, 2011) and negatively affecting both the process and the final interactive system (Moffett, 2014).

Designers and developers of interactive systems have different backgrounds and skills (Buxton, 2007) and focus on different aspects of the design process (Löwgren, 1995). Despite these differences, they need to collaborate in order to create interactive systems. However, while individuals often work closely together, their tools and artifacts do not.

This is particularly true when the design involves custom interactions and non-standard dynamic behaviors such as animations: Designers find it much easier to communicate the static visual appearance of the design than the dynamic aspects (Myers et al., 2008). By custom we mean interactions that current designer and developer tools do not provide out-of-the-box (Figure 6.1). This includes completely new interface widget as well as interactions that rely on non-standard system output, such as a highly crafted animation, or a non-standard use of user inputs, such as using a pinch gesture to delete an object.

Designers are trained to communicate visually. They use graphical editors, e.g. Bohemian Sketch or Adobe Illustrator, to create static design artifacts such as wireframes and mockups (Newman and Landay, 2000). However, common designer tools, such as vector and raster graphics, are not designed to handle interactions and animations. Designers usually prioritize visual design (Cooper, Reimann, and Cronin, 2007) over the rules and data structures that govern the software.

Complementary, developers need to create working interactions from the design specification. Developers are trained to work with abstractions, generally manipulated through textual representations. They use tools such as text editors and Integrated Development Environments (IDEs) to create functional systems. They prioritize the

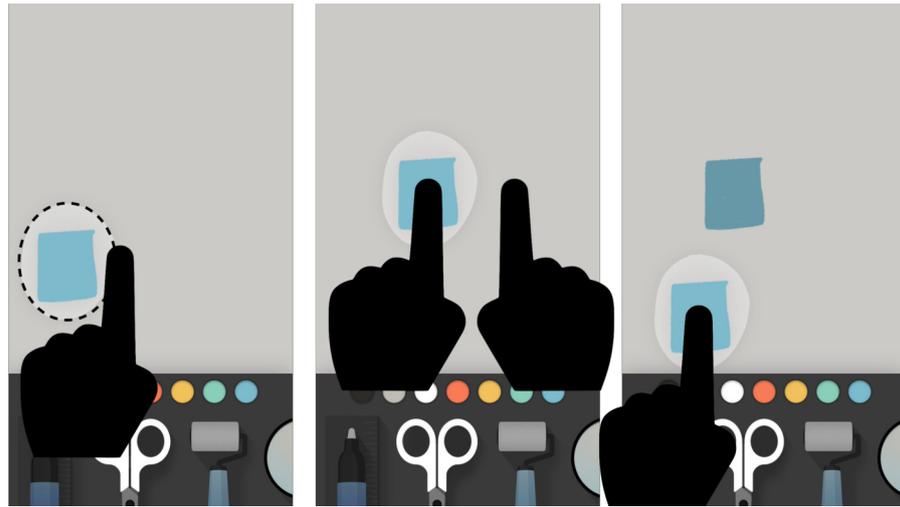


Figure 6.1: An example of a custom interaction in the Paper mobile app from FiftyThree. In the first screen, the user has selected the scissor tool and draws a circular area with one finger. In the second screen, the user drags this area to move the content to a new position and taps outside the circular area with another finger to create a copy of the selected area. In the third screen, the user drags the selected area to reveal the copied shape.

translation of design artifacts into implementable formats over the details of visual design and user interaction.

Addressing the gap between designers and developers, and more precisely the gap between the tools that they use to create interactive systems thus remains an open research question. In this work we identify problems faced by designers and developers as they collaborate, with a particular emphasis on the representation, communication and interpretation of custom interactions. Our goal is to inform the design of new collaborative prototyping tools that reduce these problems, without forcing professionals to abandon their preferred representations. We want to facilitate the transition from design to implementation, as well as the transition from implementation back to the design when changes occur. More specifically, we address the following research questions:

- What are the most common and critical problems that impair designer-developer collaboration when creating interactive software that involves custom interactions?
- How can prototyping tools help mitigate these problems?

We start by describing the motivation and methodology of the project, and review relevant related work. We then describe three studies that we conducted to better understand designer-developer practices, leading to a set of design principles to reduce the breakdowns they encounter. We introduce ENACT, a prototyping tool for

touch-based mobile interaction based on these principles, and report on two studies that we conducted to assess ENACT.

6.1 MOTIVATION AND METHODOLOGY

In this work, we define an *interaction* as the set of rules that coordinate the system outputs with the user inputs, i.e. when and how the system reacts to certain user inputs. For example, on a mobile phone, user inputs include finger touches captured by the capacitive touchscreen, user movements captured by the accelerometer, facial expressions captured by the camera, or voice commands captured by the microphone. The system outputs include both the *feedback* in the user interface, such as highlighting a button, and the corresponding *operations* to be executed by the underlying system, such as modifying a database or sending a request to a server. This paper focuses on visual outputs, but the system outputs can also include tactile or auditory feedback.

Designing, communicating and implementing interaction is complex. Fernaeus and Sundström (2012) report that “*creating a fully working interactive system is not merely a matter of ‘translating’ a static sketch into its dynamic gestalt*”. Similarly, Ozenc et al. (2010) argue that designers currently “*struggle to have a conversation with the material*”. Park, Myers, and Ko (2008) report on a laboratory study that highlights the different uses of text to represent interaction. Designer tools influenced “*their natural expression of behaviors*”, while developers used more verbose descriptions.

Designers and developers often coordinate their work through a so-called *handoff* phase. The handoff includes all the design artifacts that designers send to developers in order to construct the user interface. However, the name “handoff” is misleading, because these artifacts are not delivered and forgotten. Designers and developers refine these artifacts collaboratively and iteratively, requiring a back-and-forth work between them. This is especially true when the design includes custom interactions that lack a standard user interface vocabulary to describe them.

A number of collaborative prototyping tools have emerged to support the handoff phase¹. For example, InVision² lets designers and developers communicate with text annotations and create click-through prototypes from static design artifacts. However, developers cannot manipulate this prototype with their existing tools and the available user inputs are limited to standard discrete interactions such as button clicks. This type of tool focuses more on the information architecture of the system and the communication among practitioners, rather than on the design of custom interactions and its communication with the immaterial domain of software (Ozenc et al., 2010). Indeed, few

¹ See, e.g., <http://www.prototypingtools.co>

² <https://www.invisionapp.com/>

commercial tools let designers prototype custom interactions, usually forcing them to work with developer representations, such as code³ or visual programming⁴. Grigoreanu et al. (2009) suggest that designers needs, such as “reusing code and designs” or “automating redundant steps”, are not well supported by current software.

In order to address this gap between designers and developers, we need a better understanding of cross-disciplinary collaborative work. In their study of collaboration among different communities of practice, Star and Griesemer (1989) found two main factors that contributed to a successful collaborative work: the use of *methods standardization* and the creation of *boundary objects*.

6.1.1 *Methods standardization*

Standardization can facilitate the designer-developer collaboration by providing a shared vocabulary to describe an interaction. All major software vendors provide their own guidelines and standards for interaction, such as Google’s material design⁵ and Apple’s iOS Human Interface Guidelines⁶. Practitioners’ prototyping tools such as InVision⁷ also provide a limited and standardized interaction vocabulary to activate transitions between screens. These standards are widely used and they do facilitate the communication between designers and developers. However, they limit the vocabulary of interactions to a predefined set and therefore they cannot solve the representation issues that occur when creating custom interactions.

Another aspect of method standardization is to extend engineering representations to encompass interaction design. For example, UML⁸ diagrams have been extended to describe interactive behaviors (Silva and Paton, 2000) and to serve as a communication tool between designers and developers. But as reported by Shipman and Marshall (1999), such a level of formalism is not well received by the design community, as it does not match the types of representations used by designers.

Inspired by the popularity of software design patterns (Gamma et al., 1994), Borchers (2001) proposed the creation of catalogs of interaction design patterns, and Wiemann (2016) proposed using them as the Lingua Franca (Erickson, 2000) between designers and developers. Expanding the shared vocabulary among designers and developers can certainly facilitate collaboration. However, this language should not be limited to predefined interactions. Instead, it should enable

³ <https://framer.com>

⁴ <https://origami.design/>

⁵ <https://material.io/design/>

⁶ <https://developer.apple.com/design/human-interface-guidelines>

⁷ <https://www.invisionapp.com/>

⁸ <http://www.uml.org>

the description of custom interactions and be understandable by both designers and developers.

6.1.2 *Boundary objects*

Boundary objects (Star, 1989) are shared artifacts used by different communities of practice to satisfy the *information needs* of each of them. Star was motivated “by a desire to analyze the nature of cooperative work in the absence of consensus” (Star, 2010). Boundary objects are both flexible in collaborative use and structured in individual use. A key feature of boundary objects is *interpretive flexibility* (Orlikowski, 1992), i.e. the fact that the same boundary object can have different meanings for different communities of practice. For example, the ambiguity and incompleteness of a sketch (Buxton, 2007) provide flexibility when collaborating. The same sketch can be precise enough to describe a particular design choice when practitioners work alone.

During the design of an interaction, the most common boundary objects are *design artifacts*. For example, Newman and Landay (2000) analyzed the specificities of intermediate artifacts such as sitemaps, storyboards, mockups and prototypes. This network of boundary objects establish what Bowker and Star (2000) call a *boundary infrastructure*.

However, few studies have focused on designer-developer collaboration with respect to design artifacts. An exception is Brown, Lindgaard, and Biddle (2011) who established twelve categories of artifacts used between designers and developers. They found that regardless of the form of organization, all the designer-developer collaborations were mediated around the use of artifacts. In their study, the four most commonly used artifacts were *design ideas*, *design questions*, *interface proxies* and *stories*. However, the interactivity of these artifacts is limited or non existent, thus requiring designers and developers to envision the interaction instead of experiencing it.

6.1.3 *Our approach*

We argue that the tools used to create interactive representations, such as visual and functional prototypes, are not yet equipped to support the collaborative practices of designers and developers, especially when they create custom interactions. Therefore we seek to create new boundary objects, in the form of shared, interactive representations of the interaction being worked on. These new representations should be useful both for the collaborative process and the individual work of designers and developers.

In order to ground our work in existing practices, we started by conducting three studies to better understand the collaboration challenges of professional designers and developers. In the first study, we collected stories and identified three common designer-developer

collaboration issues. In the second study, we observed how a group of designers and developers worked around these recurring breakdowns. In the third study, we conducted a participatory design session to explore the limitations of current interaction representations.

Based on these findings, we identified four design principles to reduce designer-developer breakdowns and facilitate the collaborative construction of custom interactions. In order to demonstrate the validity of these principles, we created ENACT, a prototyping tool for touch-based mobile interaction based on these principles. ENACT features novel representations of interactions that cater to the respective needs of designers and developers within a single environment. We conducted two studies to assess ENACT: a structured observation of designers and developers using ENACT, and a study comparing the use of ENACT with traditional tools in a collaborative setting.

6.2 RELATED WORK: DESIGNER-DEVELOPER PRACTICES AND TOOLS

We grouped the related work on designer-developer collaboration in two areas. First, we review descriptive research focused on the processes and artifacts currently used during the designer-developer collaboration. Second, we review research on novel prototyping tools for creating interactive systems.

6.2.1 *Studies of designer-developer practices*

Researchers have studied collaboration among *communities of practice* (Wenger, 1998), such as designers and developers of interactive systems, from multiple perspectives. Some seek to understand the implications of combining software development processes, such as agile methodologies⁹, with design methodologies, such as user-centered design.

Although user-centered design methods for interactive systems emerged in the 1980s (Norman and Draper, 1986), their integration with software engineering processes is challenging. By the mid-1990s, Poltrock and Grudin (1994) found that large corporations' *"development practices blocked the successful application of accepted principles of interface design"*. Agile methodologies were created to address a number of software engineering issues, with an emphasis on delivering working software as quickly as possible. However, Ferreira, Sharp, and Robinson (2011) showed how this approach is often at odds with the user-centered design tradition of iterating the design from the user's perspective, before development begins. Letting developers understand what they are expected to implement, as soon as possible, remains one of the most important challenges of integrating these methodologies (Salah, Paige, and Cairns, 2014).

⁹ <http://www.agilemanifesto.org>

In the context of agile methodologies, Brown, Lindgaard, and Biddle (2011) analyzed two major aspects of the designer-developer collaboration process: collaboration events and artifacts. Their study of collaboration events shows that designers and developers constantly perform alignment work (Brown, Lindgaard, and Biddle, 2012) and that the collaboration process is organized around the use of artifacts (Brown, Lindgaard, and Biddle, 2008). However, the transition from design to implementation is not only a translation from sketches to dynamic behaviors (Fernaesus and Sundström, 2012). Most of the joint work of designers and developers is focused on managing the tensions between them. Brown, Lindgaard, and Biddle (2012) introduce the term *alignment work* as the activity of aligning designers and developers in order to seek, expose and resolve tensions. These tensions can result in *breakdowns*, which are considered to be a normal part of the collaborative work. However, even if breakdowns are commonly found in designer-developer work, we are interested in whether the limitations of the artifacts themselves are responsible or not for some of these breakdowns. Our goal is to identify current designer-developers breakdowns and propose actionable design principles that will help mitigate them.

6.2.2 Prototyping tools

Researchers have explored a number of novel tools for prototyping interactions, but most are targeted at a single community of practice rather than the collaborative work of designers and developers. Interesting approaches, such as programming by demonstration (Myers, McDaniel, and Wolber, 2000) or the use of state machines and inference engines, have not been studied in a collaborative context.

Outside the research community, practitioners have acknowledged the gap between design artifacts and their subsequent implementation, leading to a number of commercial tools. Within the past few years, over 40 commercial prototyping tools have appeared¹⁰, and a 2015 survey of 4,000 designers found that 53% of them use such tools (Vinh, 2015). These commercial tools focus on supporting remote communication between communities, assisting the extraction of design information, and helping to prototype standard and recurrent interactions. However, they do not support co-located collaboration, ignore the back-and-forth interplay between activities, and disregard the creation of non-standard interactions.

6.2.2.1 Tools for interaction developers

Code-oriented artifacts can be enhanced with other representations such as notations, diagrams and test-cases. For example, InterState (Oney,

¹⁰ <http://www.cooper.com/prototyping-tools>

Myers, and Brandt, 2014) combines constraints and state machines to facilitate reuse. InterState provides a live editor where developers can edit a program and visualize the states as they interact with the interface. Proton (Kin et al., 2012b) and Proton++ (Kin et al., 2012a) let developers use Regular Expressions to express multi-touch interactions. Juxtapose (Hartmann et al., 2008) lets developers create code alternatives and modify variables at run-time to facilitate the exploration of multiple alternatives. CodePilot (Warner and Guo, 2017) supports novice programmers by integrating coding, testing, bug reporting, and version control management into a collaborative system.

While these tools are heavily inspired by developer practices, we believe that they provide representations and enhancements that could also be suitable for designers. We are interested in how these mechanisms might be adapted to meet the needs of both audiences.

6.2.2.2 *Tools for interaction designers*

Another approach focuses on augmenting traditional design artifacts. For example, SILK (Landay and Myers, 1995) lets designers quickly create interactions using interactive sketches and envisioned “*a future in which the user interface code will be generated by user interface designers using tools*”. More than 20 years later, however, the most common approaches still require the collaboration with developers to create user interface code.

DEMAIS (Bailey, Konstan, and Carlis, 2001) provides an interactive multimedia storyboard also based on sketches. The designers’ strokes and text annotations are used as an input design vocabulary to transform static sketches into working examples. Similarly, FrameWire (Li et al., 2010) infers interaction flows from paper-prototype videos to detect hot spots and generate page-based prototypes. Forsyth and Martin (2014) use tagged digital storyboards to infer behavioral information, such as states and actions. These representations and mechanisms are closer to current practices of designers, and need to be present during the collaboration. However, we also want to bridge designer-friendly representations with more abstract representations, such as code, without excluding the participation of either community of practice.

While these tools support standard discrete interactions such as button clicks and menu selection, a few let designers prototype continuous, non-standard interactions. Monet (Li and Landay, 2005) lets designers prototype continuous widgets by demonstrating interactions on top of sketches. Designers explicitly define interaction states and the system infers the correct state through multiple examples. Using inference improves informal prototyping, but these interaction descriptions are opaque and are of limited use to the developer for the final implementation.

Other tools use intermediate representations to allow non-programmers to create interactions. EventHurdle (Kim and Nam, 2013) is a visual authoring tool for prototyping gestural interactions designed to facilitate designers' understanding and generate code automatically. While we want to provide similar mechanisms to facilitate the transition from design to development, we also want developers to be part of the prototyping process. Our work is closer to d.tools (Hartmann et al., 2006), which brings test-driven development benefits to physical prototypes. d.tools lets designers rapidly test their design and analyze results, for example to identify the most frequently used interaction. Testing is a great intersection for the design and implementation of a prototype, and it can work as a "hinge" between the two activities.

6.2.2.3 *Tools for interactive illustrators*

Recently, several authors have proposed integrating graphical and symbolic vocabularies to create dynamic graphics. Kitty (Kazi et al., 2014) is a dynamic drawing tool supporting the creation of animated scenes through functional relationships between graphical entities. Kitty relies on direct manipulation of graphics but also supports indirect manipulation of input-output functions. Apparatus (Schachman, 2017) is a graphics editor that combines direct manipulation with data-flow programming. This combination of representations, with direct and indirect manipulation, enables users to think both spatially and symbolically.

While these tools provide authoring capabilities based on various representations, they focus on the creation of dynamic drawings, illustrations and diagrams, not on prototyping interactions. For this reason, user input follows a fixed path, e.g., a constrained drag and drop, and does not take into account all the input capabilities of the target device. Victor (2013) proposes a tool that lets artists interactively create dynamic drawings with behavior simulations. The tool provides designer-friendly direct manipulation of graphics but also relies heavily on developer-friendly concepts such as linear algebra, parameterization and recursion. We are building on this trend of tools that bridge the worlds of visual output and dynamic input, but with a particular focus on the creation of interactions instead of dynamic graphics.

6.2.3 *Summary*

Designer-developer collaboration has been studied from two main perspectives: processes and artifacts. With respect to processes, practitioners and researchers both apply iterative methodologies for creating interactive systems. One of the challenges of this iterative process is to manage tensions and breakdowns in the collaboration. Some researchers see these breakdowns as a normal part of the collaborative

work. We argue that we need to better understand these collaborative tensions to determine which can be mitigated by new design artifacts.

With respect to design artifacts, most of the literature describes their use in a collaborative context, but does not provide guidelines for building new collaborative tools that reduce breakdowns. Researchers proposed myriad prototyping tools, but they have been studied with a single community of practice, either designers or developers. Involving designers and developers in the construction of interaction requires tools to support both design and development activities (Chatty et al., 2004). Such tools do not exist yet, to the best of our knowledge, especially for the collaborative creation of custom interactions.

Our goal is to create better tools that support designer-developer collaboration during the prototyping of custom interaction. To do so, we must first better understand how designers currently represent interactive behaviors to developers, as well as which aspects of these representations hinder collaboration.

STUDYING DESIGNER-DEVELOPER COLLABORATION

We conducted three studies to better understand how designers currently represent interactive behaviors to developers, as well as which aspects of these representations hinder collaboration. The following studies are presented in more detail in Maudet et al. (2017b).

7.1 STUDY ONE: UNDERSTANDING DESIGNER-DEVELOPER COLLABORATION PRACTICES

The goal of this study is to examine the existing practices of professional designers and developers in diverse settings with a particularly interested in:

- How *designers* represent and communicate a design;
- How *developers* interpret the design; and
- How *designers and developers* identify and overcome breakdowns that occur during the process.

We thus focus on understanding the difficulties that designers face when trying to express interaction to developers and the difficulties that developers face when trying to actually implement the designers' specifications.

7.1.1 *Method*

We conducted critical object interviews (Mackay, 2002) about recent design projects in order to obtain specific, detailed stories of their successes and failures targeting the collaborative aspect of the work. We focus on their most recent projects and use recent artifacts to prompt rich stories. We are particularly interested in the problems they encounter when representing and communicating interactions with each other.

7.1.1.1 *Participants*

We recruited 16 professional designers and developers (7 women, ages 24-46) from France (8), Sweden (3), Argentina (2), the USA (1), Canada (1) and China (1), who create web sites, mobile applications or interactive installations. Their work environments include: digital agency (6), design studio (4), start-up (2), freelance (2), and software factory (1).

Participants $P_{1_{ds}}-P_{8_{ds}}$ are designers (ds), self-described as UX Designer, Visual Designer, Interaction Designer, or Graphic Designer. Participants $P_{9_{dv}}-P_{16_{dv}}$ are developers (dv), self-described as Mobile Developer, Web Developer, Front-End Developer, or Creative Coder. Their experience in collaborating across disciplines, i.e. from design to development or from development to design, ranges from 1.5 to 20 years (mean 8). Half of them typically collaborate remotely and none of the participants have worked with each other. All participants reported that they were following agile methodologies during their projects.

7.1.1.2 Procedure

We interviewed participants in their studio or office for approximately 90 minutes. We asked designers to choose recent projects in which they collaborated with a developer, and developers to choose recent work in which they collaborated with a designer. For each project, we asked them to show us their tools and the specific artifacts they created, and to describe, step-by-step, the details of how they communicated the design or implementation. We probed for both successful and unsuccessful collaboration examples.

7.1.1.3 Data Collection

We collected 25 stories (one or two per participant) from different projects. During the interviews, we recorded audio and video of the participants manipulating the artifacts they created. We also photographed the final products and took notes.

7.1.2 Results and Discussion

We analyzed the 25 project stories using Thematic Analysis (Braun and Clarke, 2006). We first illustrated each project with a StoryPortrait, first introduced in (Jalal, Maudet, and Mackay, 2015), to represent the most interested stories from the interviews (Figure 7.1). A StoryPortrait includes a summarizing title, a photograph of the interactive system or a key artifact created during the project, and a top-to-bottom timeline illustrating the key steps in the collaboration process with drawings and quotes. A vertical line separates the actions of the designers from those of the developers. Arrows crossing the timeline signify a handoff between the two groups.

We studied the projects with a particular focus on breakdowns related to creating or interpreting the design documents. Then, we selected examples that formed natural categories, looking for higher-level concepts that emerged from the details of each projects. We iterated and mapped each story to one or more categories. We identified three main issues encountered by the participants during designer-

developer collaboration: reworking and redundancy, design breakdowns and late developer involvement.

7.1.2.1 *Reworking and Redundancy*

The first and most salient finding is the pervasive presence of reworking in the process and redundancy in the artifacts, not only within, but also across these communities.

DESIGNERS PRODUCE MULTIPLE DESIGN DOCUMENTS Designers communicate with developers using three primary types of artifacts: *design documents* represent the system to be implemented; *guidelines* communicate higher level information (e.g. color codes, measurements) and *corrections* describe the misalignments with the envisioned design.

All designers create multiple documents to communicate different aspects of their designs. Designers create additional design documents when the original design documents lack specificity or lead to confusion. Unfortunately, much of the information in these extra documents is redundant. Designers felt they spent too much time recreating the same information across separate documents. For example, P2_{ds} created five documents to communicate the design of a small application: UxPin “for sharing mockups”; Pixate “for detailed animations that cannot be expressed with words”; InVision “for interactive mockups with basic interactions and annotations for non-obvious features”; Photoshop because “these developers are used to work with .psd files”; and Illustrator, “the software we actually use to produce the screens.” She also used email to communicate additional design details to the development team. Unfortunately, even by using all these documents, this designer was unable to clearly communicate the design.

In addition to using *images* of “screens” to represent the visual design of the interface, designers also used video or computer-generated animations (24% of the projects) to communicate custom and dynamic effects. However, designers take more time to produce videos than static images, and developers have difficulties manipulating these dynamic representations, e.g. extracting relevant information from them. Finally, designers occasionally create *interactive mockups* (12% of the projects) using the built-in set of interactions in the tool of choice. These communicate how the interaction should feel, but only when the tool has the right set of pre-defined interactions.

DEVELOPERS RECREATE DESIGN DOCUMENTS The most common activities mentioned by the developers include interpreting the design documents and recreating them with developer tools. For example, P9_{dv} received an informal text description of a custom animation, but had to ask for a visual representation in order to fully understand the design.

We were surprised by the amount of time that developers spent recreating design documents. Some developers came up with interesting strategies to increase their productivity. One challenge is copying with different types of media: the designer's mock-ups, the code representation, and the current visual view of the system. For example, P11_{dv} created a setup with two monitors. To implement the visual design, he places the mockups on the smaller screen to assess them and he splits the other screen between a coding view and the current state of the implemented website.

Similarly, when developing a mobile radio application, P14_{dv} inserted the provided image as the background of her corresponding view in the Integrated Development Environment (IDE)'s Interface Builder. She then positioned the UI components on top of the image provided by the designer, to recreate the expected layout. She could then "*figure out the [layout] constraints*" of the screen to make it responsive, such as determining that some elements were center aligned.

Developers used different strategies to recreate the interactions described on the design documents. For example, in an interactive installation project, P10_{dv} struggled to implement the animations provided by the motion designer (Figure 7.1). He first unsuccessfully tried to use *Adobe After Effects* to extract the keyframes and curve types. Then, he asked the designer to create a text file with the needed information. This was time-consuming and boring for the designer: "*We try to set up standards, but we haven't found the right one so far*".

DEVELOPERS MISINTERPRET DESIGNS Many design decisions are lost, as developers struggle to interpret and implement the designer's original intent. In fact, none of the initial implementations exactly matched the original design. P1_{dv} felt that the developer "*used our design as an inspiration, then he made many design decisions that he did not have to make*". Similarly, P3_{ds} provided a video that showed the developer how to vary a text-box color according to the background picture. He later realized that the developer had only partially implemented his idea by sampling a single pixel, instead of generating an average color based on several pixels.

During the implementation phase, designers create *correction documents* to show the location of the mismatches and what should be modified. For example, to correct a vertical misalignment, P3_{ds} created a video. He first traced a segmented line to highlight the misalignment and then animated the correct repositioning of the elements. In the context of a real estate website project, P6_{ds} discovered several visual mismatches including wrong margins, colors and fonts. Even though he was a designer, he decided to modify the CSS code and correct the

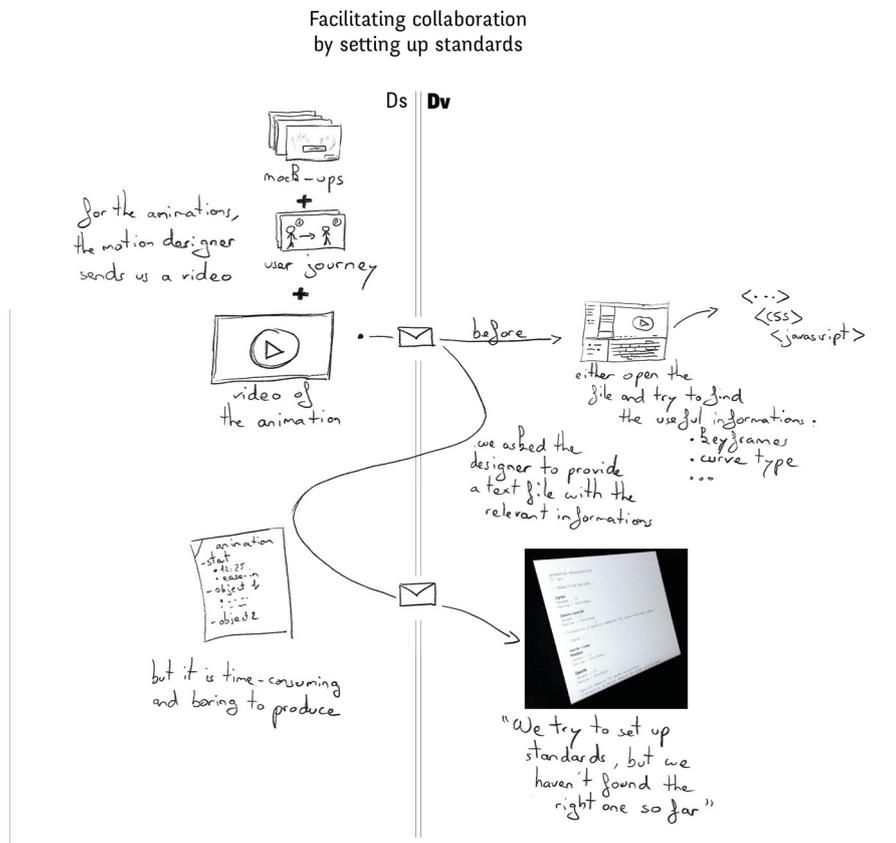


Figure 7.1: A StoryPortrait has a summary title, a photograph of a key artifact, and a top-to-bottom timeline to show the successive steps of the collaboration between designers (on the left) and developers (on the right). Participant's quotes and drawings enrich the story. Here, P10_{dv} started his story by saying that "the motion designer sends us a video". The designer sent the mockup-up, user journey and video file via email. P10_{dv} translated the video into code: "[I] try to find the useful information" in the provided artifacts. He failed to extract the relevant information about the animation, such as the keyframe timestamps and the interpolation curves. P10_{dv} finally asked the designer for a text file with all the animation parameters. The designer sent by email a the text file created by hand with all the details of the animation. Creating this file was "time-consuming and boring to produce". P10_{dv} finished his story by saying: "we try to set up standards, but we have not found the right one so far".

mistakes by himself, using the web browser Developer Tools¹. Because these changes were local to P6_{ds}'s browser, he screencaptured the new website's look and added some annotations linking the modified CSS code to the visual result. From these images, the developer then recreated all of these steps with his own tools.

STRATEGIES TO AVOID REWORK AND REDUNDANCY We found cases of rework and redundancy in all the interviews, but only two developers and one designer explicitly mentioned strategies to avoid them. P14_{dv} had a simple solution: "*Designers should just create their interfaces directly in Xcode*", referencing the Interface Builder within the IDE. P5_{ds} designed a complex casino website with many similar UI components. To avoid recreating them each time, she "*was inspired by the developers' way of working*": she created a modular styleguide that served as a shared visual library. She could then copy modules from the styleguide to create each new screen, gradually adding new modules or missing information such as the color of the hyperlinks, as requested by the developers. P12_{dv} set up a different approach. He initially received mockups and specifications for a web-based interactive advertisement builder. He built the architecture of the interface in Flash and he "*wrote the code so that the designer could very easily touch it*". The designer was able to fine tune the look (e.g. the particular images assets to be used) and feel (e.g. type of animation, time delays and duration) by directly modifying the code. This workflow helped him avoid unnecessary translation and reproduction of the designer's intentions, reducing the back and forths.

7.1.2.2 Design Breakdowns

The second main finding is the identification of three recurrent types of *design breakdowns* related exclusively to the collaboration between designers and developers (Figure 7.2). We use the term "design breakdown" to describe an impediment that must be fixed before the design can be implemented. We observed many design breakdowns such as misinterpretations, file formatting issues, mistakes introduced after modifying the original design, disagreement among the stakeholders, etc. Based on the 25 projects we analyzed, we found that the most frequent breakdowns could be categorized in three types: *missing information*, *edge cases*, and *technical constraints*.

MISSING INFORMATION A *missing information* breakdown occurs when the designer makes a decision without communicating it to the developer. Two designers and four developers reported cases of *missing information*. For example, P9_{dv} received an interactive mockup of a webpage. He could not determine whether the page's calendar widget

¹ Developer Tools are a set of tools to inspect and modify the web content of a browser window

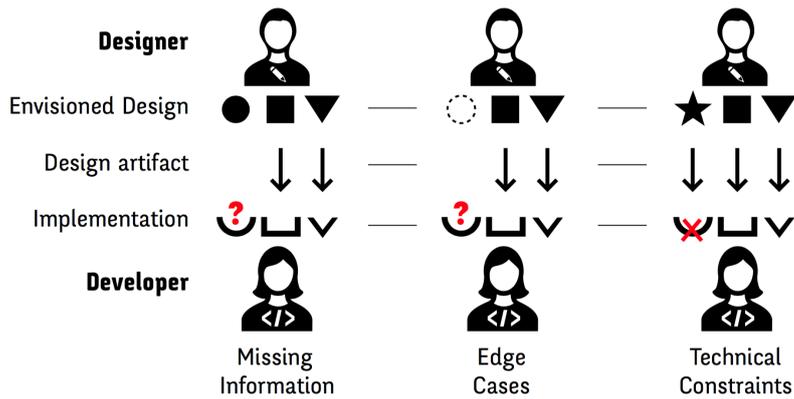


Figure 7.2: Key design breakdowns between designers and developers: *Missing information:* Designers do not communicate necessary details. *Edge Cases:* Designers do not consider certain problematic situations. *Technical constraints:* Designers are not aware of technical limitations.

was interactive or simply the output of another interaction. $P9_{dv}$ also lacked the design rationale: “What did they create that calendar for?” Similarly, $P12_{dv}$ received only static mockups for a sports application, and could not determine how to move from one screen to another.

Designers sometimes found it difficult to represent and communicate dynamic behavior to the developers. For example, $P8_{ds}$ wanted to create an animation of a blossoming flower but did not know how to represent her idea in After Effects. She ended up drawing a few sketches and then sat next to the developer as they worked out how to implement her idea directly in code.

Surprisingly, in two cases, the designers explicitly did not communicate the interaction at all, or only partially. They relied on the developer, especially because they wanted off-the-shelf interactions. This supports Myers et al. (2008)’s argument that designers find interactions hard to represent. For example, $P6_{ds}$ provided static design documents without representing some transitions, even if they were simple: “I let the developer pick the interaction between the screens, since they are very basic.”

EDGE CASES An *edge case* breakdown occurs when the designer only focuses on typical scenarios and does not consider extreme or problematic situations. Developers are trained to think about edge cases; designers are not. All developers reported that designers omit important edge cases from their design documents, and that they had to decide how to handle these situations themselves. $P13_{dv}$ received only mockups to develop a sports application. Because the designer had only specified the “sunshine cases”, $P13_{dv}$ had to make design decisions for each of the different edge cases. For example, the client asked him to include advertisements, so he modified the original

design to accommodate the ads. Similarly, P16_{dv} prepared a responsive grid for a cruise company website. The original mockup only featured the desktop version of the website. P16_{dv} did not know how to handle large elements that did not fit within the width of the screen of the mobile version: “Should the rectangle be transformed into a square or should it take a full row?” For P16_{dv}, designers usually “don’t take into account the dynamic nature of the data”.

Some designers try to overcome these issues with design guidelines. For example, P4_{ds} created a 16-page specification with annotated wireframes to explain the sign-up functionality of a website. She reported that “specifications make me think of all the states and exceptions”. She also used the guideline to capture and communicate the rationale for her design decisions. Sadly, the client’s API did not support her design and she had to modify it. Similarly, P3_{ds} created a spreadsheet to help him think and “explain the rules of the game and the limits” for each element in the website.

TECHNICAL CONSTRAINTS A *technical constraint* breakdown occurs when the designer lacks awareness of technical limitations, either in general or with respect to the developer’s skills. Five designers and four developers reported breakdowns due to such misunderstandings, which created additional work for the developer. For example, P13_{dv} received a design for an iPad application that called for horizontal scrolling when in portrait orientation. But P13_{dv} “could not recycle his code from the landscape version to create it”. He had to reimplement it from scratch, since it had already been approved by the client. This type of misunderstanding leads developers to modify the design themselves.

Lack of awareness of technical constraints is also a problem for designers. For example, when working on a complex website, the developer first told P6_{ds} that “everything was possible”. P6_{ds} soon discovered that the developer was unable to implement many elements with his tools, even though they had already been validated with the client. P6_{ds} said: “He should have said it earlier, we would have adapted our design.” Instead, they were forced to redesign the project several times to accommodate the developer’s limitations.

Collaboration is usually smoother when the designer is aware of the developer’s constraints and possibilities. For example, P5_{ds} worked on a project with two different developers. The first asked her to specify all the dimensions, such as the distances among all the elements on the screen, “so we lost a lot of time”. The second developer asked for a grid specification, which she created with 12 columns, a gutter size and a maximum size of 1200px. “Now we have the same, each one in our own tool.” The grid allowed the developer to express dimensions in percentages, sparing P5_{ds} the need to make additional annotations and saving a great deal of time.

	Developer NOT involved in the design phase	Developer involved in the design phase
Standard Interaction	<div style="display: flex; flex-wrap: wrap; gap: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P1.b</div> <div style="background-color: #c6e0b4; padding: 2px;">P5.b</div> <div style="background-color: #c6e0b4; padding: 2px;">P6.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P6.b</div> </div> <div style="display: flex; flex-wrap: wrap; gap: 5px; margin-top: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P13.b</div> <div style="background-color: #c6e0b4; padding: 2px;">P14.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P14.b</div> <div style="background-color: #c6e0b4; padding: 2px;">P16.a</div> </div>	<div style="display: flex; flex-wrap: wrap; gap: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P1.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P5.a</div> </div>
Custom Interaction	<div style="display: flex; flex-wrap: wrap; gap: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P2.a</div> <div style="border: 1px solid black; padding: 2px;">P7.a</div> <div style="border: 1px solid black; padding: 2px;">P7.b</div> <div style="border: 1px solid black; padding: 2px;">P9.a</div> </div> <div style="display: flex; flex-wrap: wrap; gap: 5px; margin-top: 5px;"> <div style="border: 1px solid black; padding: 2px;">P11.a</div> <div style="border: 1px solid black; padding: 2px;">P11.b</div> <div style="border: 1px dashed black; padding: 2px;">P13.a</div> </div>	<div style="display: flex; flex-wrap: wrap; gap: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P2.b</div> <div style="background-color: #c6e0b4; padding: 2px;">P3.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P4.a</div> <div style="border: 1px dashed black; padding: 2px;">P8.a</div> </div> <div style="display: flex; flex-wrap: wrap; gap: 5px; margin-top: 5px;"> <div style="background-color: #c6e0b4; padding: 2px;">P8.b</div> <div style="border: 1px dashed black; padding: 2px;">P10.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P12.a</div> <div style="background-color: #c6e0b4; padding: 2px;">P15.a</div> </div>

Implementation

Impossible
 Problematic
 Successful

Figure 7.3: Relationship between interaction type (standard vs. custom) and developer involvement. Lack of developer involvement in the early phase of custom interaction design is correlated with problematic or impossible implementation. Nomenclature: $P_{1.a}$ identifies the first story of participant one, while $P_{1.b}$ identifies the second story of participant one.

7.1.2.3 Late developer involvement

The third finding is that late involvement of the developer has a negative impact on the collaboration, especially when creating custom interactions. Even though all participants claimed to use agile methodologies, only five of the 25 projects (two remote and three co-located) included face-to-face sessions between designers and developers, dedicated to co-design the initial interaction. For example, $P_{4_{ds}}$ had an idea for a custom navigation rule so she drew a few ideas and invited all the designers and developers to help design it. The developer was able to implement the resulting navigation behavior without additional instructions or documents: “Nothing was written down, we only had the screens.”

Other similar examples suggest that involving developers during the design phase makes it easier to create complex interactions (Figure 7.3). In such cases, developers gain an understanding of the desired interaction during the meeting and designers need not fully represent it in the design documents. Developers were most likely to be called in for the design phase when the project included custom interactions. In most of these cases (6/8), developers successfully implemented the desired custom interaction, as in the aforementioned example of $P_{8_{ds}}$ ’s flower animation in paragraph 7.1.2.2.

However, when the project required a custom interaction and the developer was not involved at the design stage, most developers were not able to implement the proposed interaction (5/7). For example, $P_{7_{ds}}$ reported that the developer “just did not implement” the custom transition he had proposed. One of the remaining cases was still problematic: $P_{13_{dv}}$ was frustrated with the proposed interaction: “I

could not recycle my code, but as the design had already been validated by the client I still had to implement it. I lost a lot of time."

7.1.3 Summary

We identified three primary issues when designers and developers collaborate on the creation of interactive systems: reworking and redundancy, design breakdowns and late developer involvement. Both designers and developers spent too much time reworking, i.e. redoing previous work in the same or another representation, primarily due to redundancies within and across their artifacts. Designers struggle to represent interactions and dynamic behaviors with current tools and use multiple design documents to communicate different aspects of their design. Developers spend an excessive amount of time recreating the designer's documents and correcting their misinterpretations. During the implementation phase, developers face three types of design breakdowns — missing information, edge cases and technical constraints — that undermine the collaboration process. Our data also suggests that projects requiring custom interaction benefit greatly from the early involvement of the developer, during the design phase.

7.2 STUDY TWO: ANALYZING DESIGNER-DEVELOPER BREAKDOWNS

To further understand the breakdowns identified in the first study and how they are addressed, we conducted a case study of a team of designers and one developer. Unlike the first study, where interviews were based on the participants' recollection of recent projects, we observed a team during the entire duration of a one-month project. The goal of the project was to create a website for a crowd-sourced directory of companies. We were interested in whether design breakdowns still appear when a developer is involved early in the project, and, if so, which strategies are used to avoid or mitigate these breakdowns.

7.2.1 Method

7.2.1.1 Participants

We studied three designers and one developer (ages 24-25, one woman). This was the first time that this group of designers had collaborated with this developer. This grouping occurred naturally, the authors did not intervene in any of the details of the setup. One of the designers was P_{1_{ds}} from Study One.

7.2.1.2 Procedure

We observed the two face-to-face design meetings that involved all the designers and the developer. The first two-hour meeting ([Figure 7.4](#))

focused on the design of the website. The second meeting lasted an hour and focused on implementation. We also interviewed the designers separately, prior to the second meeting, to learn more about their design tools.

7.2.1.3 *Data Collection*

We video recorded both meetings and took notes. We took pictures of collaborative actions, i.e. exchanges between the designers and the developer, and their manipulation of artifacts such as drawings, notes and software.

7.2.2 *Results and Discussion*

We used Chronoviz (Fouse et al., 2011) to annotate relevant, interesting events during the meetings. Two coders marked and analyzed the times when a participant asked a question, or when a designer sought confirmation from a developer or vice versa. We correlated these annotations with the classification of design breakdowns from Study One.

We focus our analysis on the two face-to-face meetings.

7.2.2.1 *First Meeting - Accounting for design breakdowns*

The main benefit of the early face-to-face meeting was to let participants seek validation from each other and to avoid potential problems. We identified examples of avoiding *missing information*, considering *edge cases*, and clarifying *technical constraints*.

In order to avoid *missing information* (12 occurrences), the developer often encouraged the designers to specify concrete details about their design ideas. The mere presence of the developer pushed the designers to be more explicit about certain design issues. The developer also pushed the designers to think about *edge cases* (5 occurrences).

Similarly, when the designers proposed adding a gesture for deselecting a category on the mobile version, the developer asked them to consider how this design decision would affect the desktop version of the website. Given the developer's warning, the designers decided to skip the feature. Designers often sought validation, confirmation or information about *technical constraints* (17 occurrences). This echoes the "considering implementability" category observed by Brown, Lindgaard, and Biddle (2012).

In order to make informed decisions, they asked the developer about the complexity of implementing certain designs. When the designers proposed a search feature for companies, the developer asked them to specify exactly what should be searchable. The designers idea was to search within all company-related information, including their descriptions. The developer replied: "*Everything is possible... but if you*



Figure 7.4: First meeting. The developer interacts with an existing application to discuss possible interactions with two designers while another represents it on paper.

really want to make a search inside the description, it will be a bit more complex.” He suggested only looking up names and tags, but with an autocomplete feature; the designers agreed.

7.2.2.2 *Second Meeting - Fixing design breakdowns*

Even though they were able to handle many design breakdowns during the first meeting, new ones appeared during the implementation process. The developer found new *edge cases* (4 occurrences). For example, he noticed that a company card with multiple subcategories would occlude the company’s name. The designer responded: *“Maybe we can put three dots and display the extra ones only on [mouse] hover.”* The developer also requested *missing information* that he could not infer (8 occurrences). For example, when reviewing the search feature, the designer asked for a clarification: *“In which order are the items shown when they are displayed as results?”* Designers also questioned some of the developer’s decisions: *“Why do we need pagination?”* The developer proposed an alternative: *“We should [be able to] put the maximum number of items on the page without loading problems.”*

7.2.2.3 *Vocabulary mismatch between designers and developers*

In both meetings, differences in the vocabulary used by designers and developers led to miscommunication. Sometimes, designers and developers used different terms for the same concept. For example, during the second meeting the developer talked about a *“fixed”* element, referring to the CSS terminology. The designer, who tried to take the user’s perspective, referred to the same object as a *“moving”* element, an element that scrolls with the page. It took some time for them to discover that they were talking about the same behavior.

We observed several strategies for overcoming these issues (5 occurrences). Developers and designers tried to bridge the vocabulary gap by adopting each other's terminology. For example, when discussing whether an item should appear in several categories, one designer started using mathematical concepts when communicating with the developer: "Is it the union or the intersection of these two categories?" The same designer gave a specific use-oriented story to explain their decision to the other designers: "imagine if you click here [on a category] ... and then if you click here [on another category], it deselects automatically that first one." The developer also reformulated the example in terms of UI widgets: "It is either a radio button or a checkbox." On several occasions, designers and developers looked up specific interaction techniques on a particular website or found examples from a mobile application on a smartphone to show the others. This "communication-by-example" helped them verify that they were talking about the same interaction technique.

7.2.3 Summary

Involving the developer at the beginning of the design process helped the team reduce the amount of *missing information*, handle *edge cases* and set clear *technical constraints* for the scope of the design. This echoes the recommendations of Salah, Paige, and Cairns (2014) about the benefits of developer early knowledge. However, new design breakdowns occurred during the implementation phase and had to be solved collaboratively. Vocabulary mismatches also created several collaboration issues, especially when discussing interactive behavior.

Most of the work of the designers and developer was alignment work (Brown, Lindgaard, and Biddle, 2012), to manage tensions. The design breakdowns classification allowed us to analyze these tensions more finely. We found that both designers and developers actively try to mitigate design breakdowns when meeting face-to-face. However, it is still unclear which of these breakdowns are inherent to the designer-developer collaboration and which are a consequence of limitations in the design artifacts.

7.3 STUDY THREE: EXPLORING DESIGN SOLUTIONS

In order to find out whether some breakdowns could be alleviated by using appropriate representations of the design, we used a triangulation approach (Mackay and Fayard, 1997) and conducted a third, participatory design study. We were interested in whether design breakdowns are simply a natural result of the collaboration process, or if they are also by-products of the limitations in the representations used to describe interactive systems. Since these representations are traditionally the product of designers, we wanted to elicit new kinds

of representations by asking designers and developers to create them together.

7.3.0.1 *Participants*

We recruited two designers and two developers (all men, ages 24-33). Two of them (one designer and one developer) participated in Study One and were invited due to their interest in this research. The other two professionals were recommended by participants from the previous studies. The developers had not previously worked with the designers. They had 1.5 to 10 years of experience collaborating across disciplines. Besides the four active participants, the authors of this paper attended the workshop: two as observers and two as participant-observers.

7.3.0.2 *Procedure*

The workshop lasted three hours and featured two activities designed to examine how designers represent and communicate *existing* custom interaction behaviors. We selected two unfamiliar interaction techniques from two mobile applications that rely heavily on continuous gestures. Participants were given the opportunity to explore these techniques for themselves on a mobile device we provided. The techniques were:

- *Pinch-to-create*: The Clear to-do list mobile app² uses a pinch out gesture to progressively split apart two items and create a new one between them; lifting the fingers creates the item (Figure 7.5).
- *Pan-and-stamp*: The Paper note-taking mobile application³ uses a lasso technique to select an area of the canvas to be cut, which can then be moved with a panning gesture. While moving it, a tap with another finger copies it at that particular location.
- *Lasso-fill*: The Paper app also uses a lasso selection to specify an area to fill with a color previously selected. When the lasso crosses itself, the area is colored with the so-called even-odd winding rule, leading to unexpected results (Figure 7.7a shows a sketch of the interaction).

Activity 1 (1h): Designers and developers were divided into two pairs, grouped by roles. Designers received *pinch-to-create* and developers received *pan-and-stamp*. We made sure that none of the participants knew these interactions beforehand. We asked the designers to describe the interaction as they would ideally communicate it and asked the developers how they would like to receive a description from a

² <http://www.realmacsoftware.com/clear/>

³ <http://www.fiftythree.com/paper>

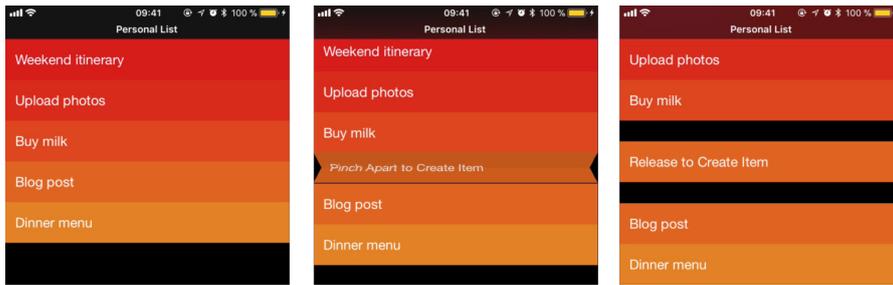


Figure 7.5: The *pinch-to-create* interaction is based on the Clear to-do list mobile app. First the user puts down two fingers simultaneously. Then, by spreading them, the new item is revealed progressively. Finally, the new item is created when the user lifts her fingers off the screen.

designer. We asked them to give as complete a description as possible, and gave them access to all the tools and means they use in their daily work practices. When participants were satisfied with their representation, they gave the resulting artifacts to the other pair. Each pair then tried to describe what they understood from the representation. We also asked them to try to find the interaction in the real application. Afterwards, participants discussed the issues they encountered when creating and interpreting the representations.

Activity 2 (2h): The two designer-developer pairs were shown the *lasso-fill* interaction. We asked each pair to come up with strategies or new representations that fully communicate the original interaction. We asked them to create representations that satisfy both members of the pair.

7.3.0.3 Data Collection

We collected all artifacts created by the participants: sketches, diagrams, text descriptions, paper prototypes and stop-motion videos. We took photographs and videos as they manipulated these artifacts, and took notes during the discussions.

7.3.1 Results and Discussion

The workshop provides clear evidence that current design artifacts are not adequate to provide complete descriptions and avoid design breakdowns. Designers and developers had to collaboratively combine different types of representations, such as sketches and diagrams, to create complete and general descriptions from concrete and specific examples.

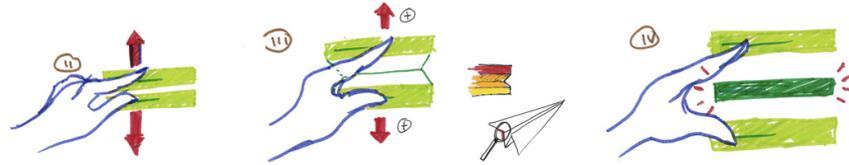


Figure 7.6: An example of a designer’s representation of *pinch-to-create*. The designer depicts a continuous interaction by discretizing key visual states and adding user input annotations.

7.3.1.1 Current representations do not encourage completeness

Participants took approximately 15 minutes each to create and be satisfied with their representations in the first activity. Even though they were given a fully functioning interaction and instructed to create complete descriptions, the four proposed representations were clearly incomplete. This suggests that some design breakdowns are a by-product of inadequate representations.

The designers relied primarily on visual representations based on drawings and annotations. Developers felt that these were effective in communicating the overall idea, but left too many unanswered questions for a correct implementation. For example, the designers did not communicate certain types of feedback, such as the gesture spread threshold or the animated transition that placed new items at the top of the list. During the discussion, one of the developers explained that a picture requires more translation steps than a text description for implementation: “If I receive a picture, I first need to translate it into text and then I need to translate it into code.”

Developers relied mainly on text, including programming vocabulary such as conditionals and loops, complemented by a few visual elements. Text descriptions provided specific information for the implementation but did not clearly convey the look and feel of the interaction to the designers. For example, when trying to represent the *pan-and-stamp* interaction, developers did not communicate the increased opacity outside the selection and the flash effect when pasting the copied area.

7.3.1.2 Strategies for creating complete representations

During activity 2, the two pairs explored seven different strategies to fully represent and communicate the interaction. The most promising strategies were:

- A. Pair 1 decomposed the *lasso-fill* interaction using examples from other applications: the lasso tool from Photoshop combined with the paint bucket from Illustrator. The designer from Pair 1 proposed recording videos to demonstrate the use of these tools

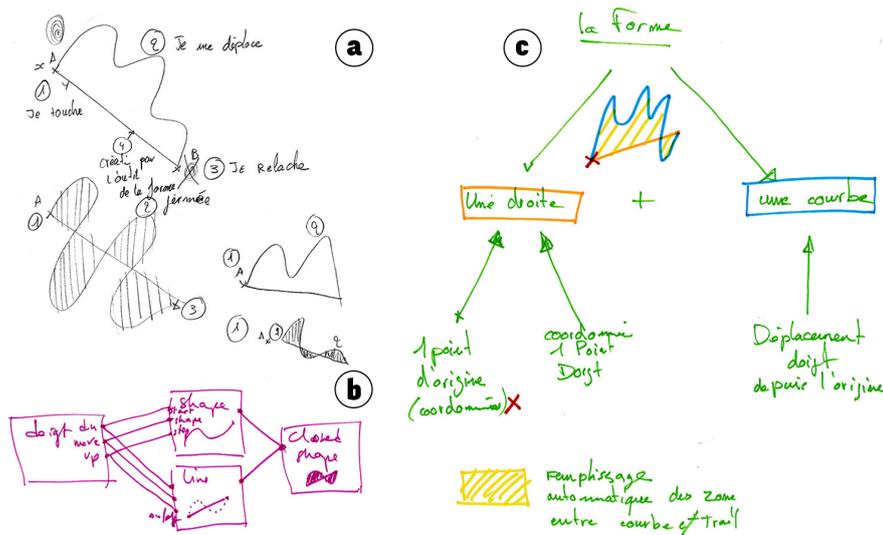


Figure 7.7: (a) A designer drew a snapshot of the lasso-fill interaction at four points in time. (b) A developer created a diagram connecting primitive graphical elements and functions with user inputs. (c) The designer merged the two representations with an example.

and combine them. He argued that this strategy would avoid misunderstandings as well as provide a complete description of the interaction.

- B. The developer from Pair 1 proposed a shared “lexicon” describing the objects of the program, their characteristics and the tools that can interact with them. Pair 1 thought that a common vocabulary would facilitate the discussion about how to extract common components.

In order to reach a shared and complete representation, both pairs refined their representations through multiple iterations. They started with a visual example, and then added rules and annotations to produce a more complete description. For example, the designer from Pair 1 drew a snapshot of the interaction at four points in time: touch, move, release, and create closed shape (Figure 7.7a). Next, the developers and designers collaborated to gradually generalize the description of the interaction. The developer, inspired by the designer’s representation and his knowledge of “flow programming”, drew a diagram representing the different components of the interaction (Figure 7.7b): finger, shape, line and closed shape. Based on the developer’s representation, the designer built a new representation that combined the strengths of both proposals (Figure 7.7c). He color-coded a visual example and mapped each graphical element to a detailed description of the expected behavior.

7.3.2 Summary

Even when provided with an existing and complete interaction, both designers' and developers' representations suffered from *missing information* and *edge cases*. This suggests that current representations are limited and may result in design breakdowns. To create complete representations, designers and developers gradually added rules in addition to concrete examples. This way of working helped designers and developers to mitigate breakdowns. We argue that computer-based prototyping tools supporting this workflow between designers and developers could also mitigate collaborative breakdowns.

These three studies show that breakdowns occur routinely during designer-developer collaboration and can result in severe mismatches between the system originally envisioned by the designers and its final implementation. To avoid such consequences, we should help designers and developers identify these breakdowns and solve them as early as possible.

A key source of designer-developer breakdowns is the lack of an infrastructure supporting the *boundary objects* used to represent interaction. Designer tools generate isolated representations that are rarely updated when the system implementation changes. Designers are also responsible for manually keeping the design consistent across multiple design documents, often in different formats, e.g., images, diagrams and video. By contrast, developer tools work with precise and “executable” representations. These representations govern the actual system and developers must ensure that they are up-to-date with the expected design. In such a workflow, when changes emerge during iterations, the design specification is rarely materialized outside of the final implementation due to the high costs of updating multiple and unconnected representations. We believe that tools for the design and implementation of interactive systems must accommodate the skills, values and practices of both designers and developers, and support an integrated workflow.

8.1 DESIGN PRINCIPLES FOR DESIGNER-DEVELOPER COLLABORATIVE TOOLS

Based on these findings, we propose four principles for the design of computer-based tools that support designer-developer collaborative prototyping and reduces breakdowns:

1. Provide multiple viewpoints;
2. Maintain a single source of truth;
3. Reveal the invisible; and
4. Support design by enactment.

These design principles address the following questions:

- How can we encourage early participation by developers?
- How can we reduce reworking and redundancy?

- How can we mitigate design breakdowns?
- How can we support a workflow from example-based to rule-based descriptions?

8.1.1 *Principle One: Provide multiple viewpoints*

A collaborative prototyping tool should provide multiple representations of the interaction in order to support the different viewpoints of each community of practice involved in the process.

In order to encourage *early developer involvement* in the design process, we need to provide a designer-developer "sandbox" that supports both designer- and developer-friendly representations for manipulating the system under construction. In other words, if we want designers and developers to, e.g., "sketch" an interaction together, we should embrace the needs and perspectives of both communities. In the following, we use the word *viewpoint* to characterize a representation of the design targeted to a particular community of practice or activity.

Today's design and development practices do not encourage the use of multiple viewpoints concurrently. Our studies show that in the early stages of a project, tools tend to focus on rapid prototyping, disregarding the developer's view; in later stages, tools tend to focus on development, disregarding the designer's view. From the start and throughout the entire design process, designers and developers should be able to manipulate the interactive system under construction with their own representations and tools. By providing multiple viewpoints, we can leverage the existing skills and knowledge of both designers and developers.

These viewpoints should provide rich interactions to manipulate the design, similar to those in the current individual tools. For constructing the user interface, IDEs such as Eclipse or Xcode feature a user interface builder as their "design view" and a code editor as their "code view". However, they provide limited authoring capabilities compared to existing vector and raster graphics software. UI builders focus on assembling predefined elements, dragged from a catalog of interface components, rather than creating or exploring custom interactions. These "design views" are used differently than graphical authoring tools; designers can only assemble canned interfaces within a constrained environment, while graphical authoring tools let them explore new ideas in an unrestricted environment.

During the workshop, designers and developers started with specific, concrete examples, and then, through several iterations, generalized these examples to create more abstract representations. This suggests that designers and developers would benefit from *viewpoints at different levels of abstraction*: some more concrete to represent exam-

ples and some more abstract to describe rules. We believe that the ability to transition from specific examples to higher-level abstractions would help designers create more complete descriptions.

8.1.2 *Principle Two: Maintain a single source of truth*

A collaborative prototyping tool should keep the multiple representations of the interaction synchronized, i.e. changes in one representation should be reflected in the others as quickly as possible.

In order to reduce *rework* and *redundancy*, prototyping tools should not only provide multiple viewpoints, but also keep these representations in sync. The previous studies showed the fluid and iterative nature of designer-developer work, and the need to modify the representations during the process (P_{4ds} , P_{13dv} , P_{16dv}). Some graphical tools support “symbols” or “smart objects” that are referenced across documents instead of being copied. Designers can modify these smart objects and see the changes reflected wherever they are used, alleviating the need for manual synchronization. However, while this encourages modularity it also requires planning: “smart objects” must be created before being used. Such planning is at odds with the fluidity of the design and implementation process.

We argue that design tools should support objects that are consistent across representations, minimizing the need for planning. To foster collaboration and understanding across activities, changes in one representation should be instantly reflected in the others, similar to live programming (Tanimoto, 2013). One strategy is to use automatic transformations among representations. For example and when possible, changes to a designer-friendly representation would be made available to implementation representations and vice versa. This approach encourages reuse across activities and reduces redundancy by leveraging shared entities.

In practice however, some information might not be transformable among representations. For example, a representation focused on the *look* might not be able to represent the *behavior* of an interaction. Also, it is usually impossible to transform among representations working at different levels of abstractions. In such cases, users should be notified that changes have been made, even if they cannot be automatically transformed into the target representation. In addition, the system should help users regain a synchronized state.

8.1.3 *Principle Three: Reveal the invisible*

A collaborative prototyping tool should reveal information only available in one representation in the other representations; when not possible, intermediate representations should be provided.

In order to reduce cases of *missing information* it is not enough to maintain multiple viewpoints synchronized. Unlike in current workflows, where some elements are only visible and manipulable in certain representations, we need to augment or create new intermediate representations to expose information that is not available in existing design artifacts.

In our studies, developers often lacked access to important elements of the design such as measures (P_{5ds}), colors (P_{6ds}) or animation time values (P_{7ds}). Some of these elements were available in design documents, but developers did not have access to them and had to ask the designer to send them explicitly (P_{5ds}) or in a specific format (P_{2ds}). In some cases, the visual elements used to communicate extra information, such as annotations, measures or user inputs, were misinterpreted as actual visual elements that were part of the user interface. For example, in the second study the developer misinterpreted a cross in the corner of a widget for an annotation, when in fact it was a close button. Prototyping tools should reveal this design information to the developers without adding noise to the design being communicated.

Conversely, designers lacked access to information hidden in the code. For example, measures or equations used as parameters of an interaction exist in code but do not have a manipulable visual counterpart in the design representations, therefore designers cannot modify them by themselves. Even worse, designers needed to produce extra documents to communicate the appropriate request to the developer. Prototyping tools should reveal these interaction parameters to the designers in a form compatible with their viewpoints.

Ideally, all entities and concepts should be available in all representations. When this is not possible, hints or links should be made explicit to understand these invisible relationships among representations. For example, *missing information* could be more easily detected when a design decision is not available on the developer's view. Developer concepts such as parameters and conditions should be manipulable by the designer, especially if they are linked to concrete aspects of the design, such as distances, relative positions or extreme values.

8.1.4 *Principle Four: Support design by enaction*

A collaborative prototyping tool should support the design and implementation of an interaction through enactments of the interaction as a rapid, active and contextualized medium for design.

In order to find *edge cases*, the prototyping environment should let designers and developers experience the interaction being created in context as early as possible. In the same way as we informally communicate an interaction idea by gesturing with our hands, computer-based tools should take as inputs enactments of the interaction performed with our own body, including hand gestures, voice commands or gaze.

According to Bruner:

“Any domain of knowledge [...] can be represented in three ways: by a set of actions appropriate for achieving a certain result (enactive representation); by a set of summary images or graphics that stand for a concept without defining it fully (iconic representation); and by a set of symbolic or logical propositions drawn from a symbolic system that is governed by rules or laws for forming and transforming propositions (symbolic representation)”

—Bruner 2006.

Victor (2013) calls these three modes of representation *interactive* (enactive), *visual* (iconic) and *symbolic*. In our studies, we observed that designers create visual representations of interactions and dynamic behaviors as screen flows with different levels of precision, such as sketches, wireframes or mockups. Designers typically describe how the system reacts to an imprecisely specified user input. Based on these designs, developers create symbolic representations of the interaction in the form of code that they run in the target environment, or in an emulator. Developers then assess the extent to which the implementation follows the specified design.

This process restricts the use of interactive devices for testing, rather than a full-fledge design medium. We argue that interactivity should be used during the entire design and development process, not only at the testing stage. For example, in the second study, the developer interacted with existing mobile applications to show different design possibilities to the designers. In the first study, P_{4,ds} organized a co-creation session so that she and the developers could “act” over some drawings to show how the interaction should behave.

User inputs, not just visual outputs, should be manipulable. They should be created in context on the target environment to inform the design and encourage “learning by doing”. Also, the relationships created between user inputs and visual outputs should be “tweakable” and explorable. In all three studies, we observed that designers were less likely than developers to specify *edge cases*, even though their early identification can avoid significant problems later on. However, they paid more attention to *edge cases* during the workshop, when designers and developers were encouraged to work with the target device.

Current practices do not encourage enaction: designers typically work during the entire process with graphical editors and do not

explore their design on the final device until an implementation starts working. *Supporting design by enaction* lets creators play an active role as the final user; finding *edge cases* becomes part of the construction itself.

8.1.5 Summary

The goal of these four principles is to minimize designer-developer breakdowns due to the limitations of their design artifacts:

- *Provide multiple viewpoints* of the designers and developers needs and practices, to help developers participate early in the design process;
- *Maintain a single source of truth* to keep all these viewpoints in sync and reduce *reworking* and *redundancies*;
- *Reveal the invisible* information, either hidden from the designers in the code or sheltered from the developers in the design artifacts, to avoid *missing information* in the design descriptions; and
- *Design by enaction* to find more *edge cases* by experiencing the design in context as soon as possible.

These design principles are also tightly interconnected. Each principle targets a particular breakdown but can also help mitigate other breakdowns. For example, *providing multiple viewpoints* encourages the early participation of developers, and should also reduce *missing information*. *Maintaining a single source of truth* reduces reworking and redundancy, which should also facilitate early developer participation. *Revealing the invisible* reduces *missing information*, and can also make *edge case* more visible. Finally, *supporting design by enaction* can help find *edge cases*, but can also reveal *technical constraints* by reducing the time-to-interaction.

Taken together, these principles lead to the definition of a *network* of inter-related design artifacts for prototyping interactive systems, instead of the current situation where the design artifacts are loosely connected. However, we acknowledge that applying these principles is not sufficient to guarantee successful collaborative work. Indeed, some designer-developer breakdowns are not directly related to the artifacts that they use: Interpersonal relationships, different styles of communication and technical knowledge, just to mention a few, also have an important impact on designer-developer collaboration. Nevertheless, these design principles, which are grounded in our in-depth studies, can help researchers and practitioners create new prototyping tools that significantly reduce avoidable breakdowns, as we now illustrate.

8.2 ENACT: A TOOL FOR COLLABORATIVE PROTOTYPING OF TOUCH-BASED INTERACTIONS

In order to validate the above design principles, we need to use them in real collaborative scenarios. One option is to start with existing designer and developer tools, but this requires integrating proprietary software from different vendors and potentially incompatible data representations. Instead, we chose to create a new tool, called ENACT, which gave us more freedom to explore the use of the design principles in an unrestricted environment.

8.2.1 Overview

ENACT is a live environment for prototyping custom touch-based interactions that supports collaboration between designers and developers. Among the many interaction styles that can benefit from collaborative support, we decided to focus on multi-touch interaction for mobile devices. As we have observed in the participatory workshop, multi-touch interactions are familiar yet challenging for professionals to describe and prototype. ENACT lets users work graphically and with concrete examples, but also exposes low-level touch events through an interactive state machine.

There are many aspects of the construction of an interactive system that are outside the scope of ENACT. Our focus is on supporting designer-developer collaboration when creating custom interactions rather than supporting the creation of a complete interactive system. Therefore ENACT does not cover back-end integration for networking, storage, application distribution, etc. It also does not have means to specify the navigation among multiple screens. While ENACT could be extended to support these features and allow the creation of full-

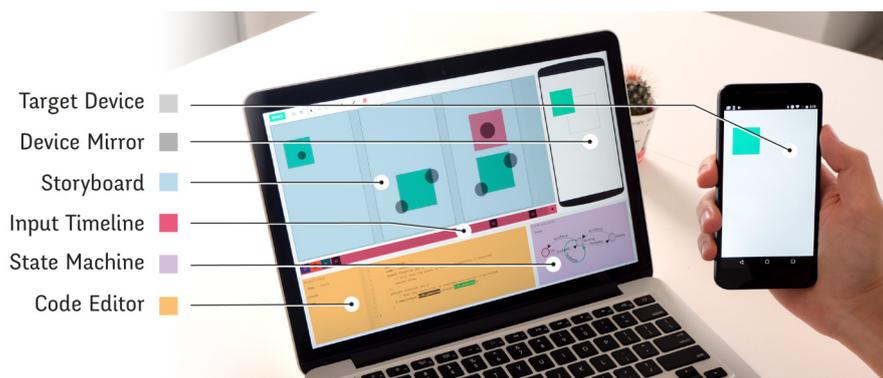


Figure 8.1: ENACT uses a target mobile device and a desktop interface with five areas: a storyboard with consecutive screens, an event timeline with a handle for each screen, a state machine, a code editor and a device mirror

fledged touch-based interfaces, the current version only supports the definition of a single interaction on a single screen.

ENACT addresses the main collaboration issues identified in the first three studies by applying the four previously described design principles:

- The desktop interface is organized in five areas providing *multiple viewpoints* of the interaction (Figure 8.1): a storyboard, an input timeline, a state machine, a code editor and a device mirror;
- The *single source of truth* is the interaction under construction, maintained automatically across viewpoints or manually with the assisted testing;
- The device mirror *reveals invisible* information such as measures and touches; and
- The target device supports *design by enaction* by letting users perform gestures and test the interaction in real time.

We first illustrate ENACT through a use scenario. Then, we describe the main features of ENACT and provide their design rationale.

8.2.2 Use scenario

Anton, a designer, and Petra, a developer, want to prototype a custom interaction to create items in a to-do list application using a continuous spread gesture, or “pinch apart”, i.e. the reverse of a pinch gesture (Figure 7.5).

8.2.2.1 Representing the visual design

To communicate the visual design to Petra, Anton first draws the *look* of the interface at different stages, similar to the sketches presented in the third study (Figure 7.6).

Anton starts by drawing the to-do items as rectangles in the first screen of the storyboard. When Anton adds a second screen, ENACT duplicates all the elements from the previous screen, a feature called storyboard automatic propagation (Figure 8.2a). In the second screen, Anton adds a 6-vertex polygon as the new item and adjusts its size to fill the space between the two existing items. He then creates a third screen where he moves the items apart and increases the size of the new item in between by modifying the vertices. Finally, Anton decides to tweak the color and the size of the first two items. He only needs to modify these in the first screen because changes automatically propagate to the following screens. This propagation reduces redundancies and ensures consistency across the design.

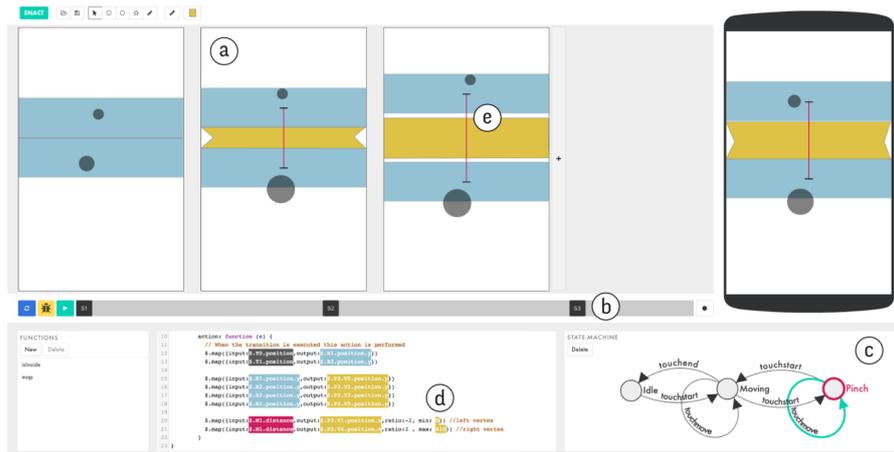


Figure 8.2: Scenario. The first screen of the storyboard shows the initial look of the interaction. The two items created in the first screen (light blue) propagate to the second screen, where a third item (in yellow) is added (a). Each screen has a corresponding handle in the input timeline (b). The state machine shows the selected transition in green and the active state in red (c). State machine actions can be edited in the code editor (d), either by coding or by linking a storyboard elements to create *design references*. In the third screen of the storyboard, a measure M1 is added between the top and bottom rectangle (e).

8.2.2.2 Specifying user inputs

To communicate the interaction design to Petra, Anton records a spread gesture directly on the mobile device, mimicking how the user should interact with the system (Figure 8.3). The recorded input events are stored in the timeline which features a handle for each screen in the storyboard (labelled S1, S2 and S3 in Figure 8.2b). Anton can move the screen handles along the timeline to position the screen at the moment in time that corresponds to the right distance between the finger touches of the recorded gesture. He can also adjust the items in screen S2 to better match the gesture. Since these items are in a different position in screen S3, the automatic propagation does not apply. By interacting with the device and by seeing the user inputs on top of the screens, Anton can make informed decisions about the layout of the interface with respect to the position of the user touches.

8.2.2.3 Making the design interactive

Petra receives Anton's design and examines the screens. ENACT automatically generates an animation combining the storyboard screens and the recorded user inputs. She watches the animation to better understand the behavior of the interface.

ENACT uses a state machine (Appert and Beaudouin-Lafon, 2006; Oney, Myers, and Brandt, 2014) to specify the interaction behavior. When creating a new interaction, a default state machine is provided.

As Petra interacts with the target device, she can visualize the transitions among states in real time. She decides to add a new Pinch state to create the custom pinch apart gesture (Figure 8.2c). During the pinch, the user touches should control the position of the top and bottom items. To do so, Petra selects the touchmove transition of the Pinch state and programs a new action in the code editor (Figure 8.2d). She can write code but she can also drag touches, visual elements, and properties from the storyboard to automatically insert a reference to these objects, called a *design reference*. After adding this new action, Petra can directly test the interaction on the mobile device.

8.2.2.4 Revisiting the interaction

Petra notices that the new item, in the middle of the other two, is not being resized. To control the positions of the new item's vertices, she needs to know the distance between the top and bottom items. Instead of writing an equation in the code editor, Petra creates a *measure* (M1) in the storyboard: She drags a line between the two original items with the measurement tool (Figure 8.2e). Now, she can drag the new measure into the code to create two *mappings* between M1's distance and the positions of the new item's vertices. She is satisfied with the result and sends the design back to Anton.

Anton runs the assisted test, i.e. an automatic comparison of the storyboard elements with the result obtained after running the code with the recorded inputs, and notices that the new to-do list item grows beyond the width of the previous items. Anton decides to fix the implementation himself. He adds a maximum value to the second mapping created by Petra. Anton drags the values from the third screen, and drops them as maximum values in the code (Figure 8.2d). Since these values are now bound to screen elements, Anton can tweak the size directly in the storyboard instead of modifying the code by hand.

8.2.3 Drawing the user interface

The initial studies showed that designers generally depict the different visual states of the interaction with sketches (Figure 7.6), diagrams, wireframes or mockups. In the first study, we found that designers needed to maintain consistency among screens manually, e.g. with copy-paste, therefore introducing redundancy in the artworks that later turned into mismatches or inconsistencies.

In ENACT's storyboard, each screen is aware of its past and future screens (Figure 8.2a). Following the *maintain a single source of truth* principle and in contrast to "smart objects", ENACT's propagation mechanism is active by default. Whenever a designer creates a new shape or changes a property of an object, that change propagates to all future states. Creating a new screen automatically imports the shapes

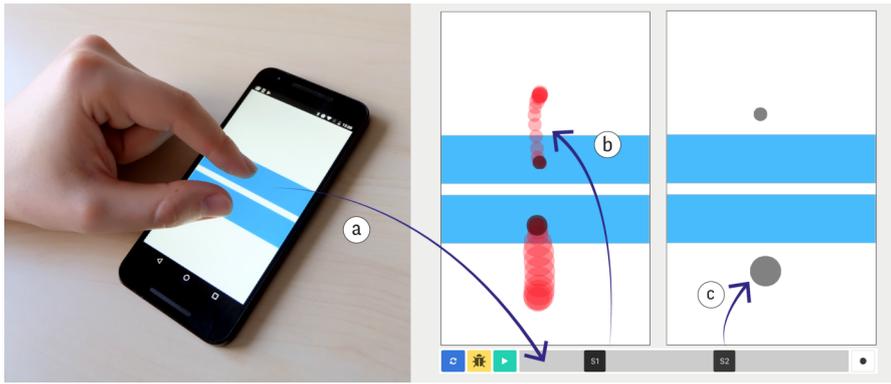


Figure 8.3: Recording an input on the target device. The input events are saved in the timeline (a), the designer can navigate the recorded inputs by dragging them or by moving the associated screen handle in the timeline (b). Current touches are displayed as translucent grey ellipses on each screen, the radius of the ellipse represents the size of the touch (c).

and objects from the previous screen. Changes include transformations, such as moving, resizing, or changing colors. A change in, e.g., Screen₂ propagates to subsequent screens (Screen_{3,4,...}), but also stops propagation from Screen₁. To reactivate propagation for an object, the user simply needs to modify it in the screen where it was changed to match the same object in the previous screen. Propagation reduces the amount of redundant information across screens, without modifying current designers' practices.

8.2.4 Providing concrete input examples

The first study showed how designers generally describe user input as annotations on top of the visual design. Following the *support design by enaction* principle, we wanted to generate this information on the fly from actual interactions rather than using static annotations drawn by the designer. Unlike current design tools, ENACT lets designers associate screens with actual user input events, recorded directly on the target device. This approach is similar to “programming with examples” (Kato, Igarashi, and Goto, 2016). First, the designer presses the record button at the right of the input timeline (Figure 8.3). While the designer performs the desired input on the target device, real-time feedback appears on the device mirror. Once all touches have ended, the recorded input events are saved in the timeline and associated with the storyboard's screens.

Touch events are treated as first-class objects: They can be displayed and manipulated like other graphical elements. This builds on the existing design practices of annotating sketches with gestures, but also provides new capabilities. For example, touch inputs can be used to position other graphical objects or to compare relative sizes. This

helps designers better understand the *technical constraints* of designing for mobile, e.g. to determine if an object is big enough to be touched reliably.

8.2.5 *Generating an animation from the storyboard*

The first study showed that designers rely on animations or videos to describe custom interactions. Animation is a simple medium for illustrating an interaction, but designers currently need specialized tools to create them. In order to *provide multiple viewpoints*, ENACT automatically generates animation descriptions based on the storyboard and the recorded input example. Therefore, designers can create simple animations in ENACT by reusing the storyboard instead of creating extra documents.

Since each screen is associated with an input event, ENACT knows the time between screens and can therefore animate the visual changes by using the screens as keyframes. The current implementation uses linear interpolation, but other interpolation functions could be added.

After pressing the play button, the animation is executed on the target device and the mirror, displaying both the visual objects, the touch events and any created measure. These lightweight animations let designers check the timing between the recorded touch input and the screens in the context of a real device. While these animations are not interactive, they provide a stepping stone towards creating an actual interaction, which requires programming mappings between user inputs and screen outputs.

8.2.6 *Programming interaction*

The initial studies showed that the traditional dichotomy between “design view” and “code view” is not enough to articulate designer-developer collaborative work. Rather than forcing designers and developers to abandon their practices, we want to augment them. Developers frequently use diagrams to describe an interaction and think in terms of states. However, this is rarely reflected in the tools they use. ENACT uses an interactive state machine to represent the code of an interaction. The state machine *reveals invisible* details that are usually buried in the source code. It also organizes the interaction code as a sequence of transitions that occur over time, as opposed to the traditional set of event handlers organized by event type.

ENACT also leverages the visual representations to support programming. Developers can use *design references* to connect interface elements or specific values between the storyboard and the code. They can also create *dynamic measures* by direct manipulation instead of defining symbolic formulas in the code.

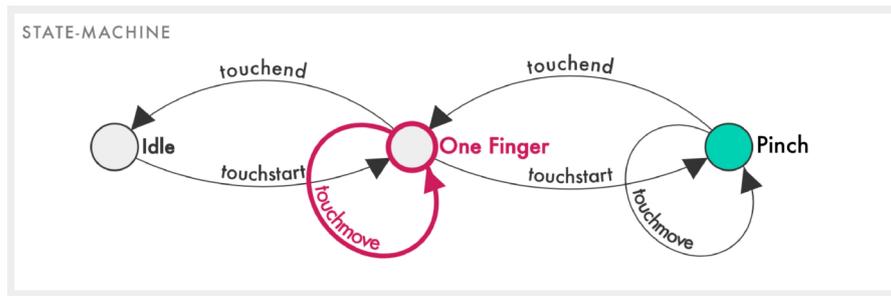


Figure 8.4: A state machine for a pinch interaction. The state machine diagram is interactive: states and transitions can be added, removed or edited. The selected state, in green, can be edited in the code editor. The currently active state and transition are highlighted in red.

8.2.6.1 State Machine

The goal of ENACT is to let designers and developers create custom touch-based interactions. Implementing such interactions typically requires processing low-level touch events, which is tedious and error-prone (Kin, 2012). Instead, ENACT organizes the interaction code around a state machine that exposes touch events as transitions (Figure 8.4).

Previous work has demonstrated the power of state machines to describe interactions (Appert and Beaudouin-Lafon, 2006; Oney, Myers, and Brandt, 2014). State machines also *provide multiple viewpoints*: the high-level logic of the state machine, described by its graph, and the low-level details of transitions and actions, programmed in code. These viewpoints provide representations that both designers and developers can manipulate. State machines also gather an entire interaction within a single object, providing a single *source of truth*. Finally, ENACT highlights the active state and transition in real time when interacting with the mobile device, *revealing the invisible* inner working of the interaction.

ENACT provides a default state machine, with two states and three transitions, that supports continuous touches with one finger. To add multi-touch capabilities, users can create new states by double-clicking an empty space in the diagram and new transitions by control-dragging from the source state to the target state. When the user selects a state or transition, ENACT shows its code in the editor as a plain JavaScript object (Figure 8.5). Transitions are named after their input event, they have a source and a target state (extracted from the diagram), an optional guard and an action. ENACT only executes the transition's action when its input event is detected and the guard is satisfied. States can also execute actions when activated (on enter) or deactivated (on exit).

ENACT is a live environment: guards and actions are written in JavaScript and interpreted right away. Because there is no waiting time,

```

1 {
2   description: '',
3   name: 'touchmove',
4   guard: function (e) {
5     // Only when the guard is true the transition is executed
6     return $.isInside({touch:$.T0,shape:$.R1});
7   },
8   action: function (e) {
9     // When the transition is executed this action is performed
10    $.map({input:$.T0.position, // position of a touch
11          output:$.R1.position, // position of a rectangle
12              max:237}); // y-position of a rectangle in the first screen ($.S1.R1.position.y)
13    }
14 }

```

Figure 8.5: The editor shows the code of the selected state or transition. Here, a touchmove transition is selected. The guard ensures that the transition triggers only when touch T0 is inside rectangle R1. In the action, the developer has defined a mapping between the positions of the touch and the rectangle. The editor creates *design references* around recognized design elements: \$.T0.position, \$.R1.position and \$.S1.R1.position.y. The number 237 is a *local design reference* representing the y-position of rectangle R1 in screen S1. If the storyboard changes, this value will also change.

guards and actions can be edited live and the result is immediately available. As a result, users can *design by enactment* by immediately testing the current design in context.

8.2.6.2 Reusing design elements with design references

For developers, the storyboard not only describes the interaction, it also provides a repertoire of visual objects ready to be used in the code. Developers can drag elements from the storyboard to the code editor to create *design references* (Figure 8.2d), i.e. code-based representations of the visual elements manipulated by designers. To help match the code representations with its visual counterpart, they share the background color of their linked visual element (Figure 8.5). Developers can modify them directly with a double-click in the code editor.

When the developer hovers over a *design reference*, the storyboard highlights the corresponding element. As an alternative to dragging the visual element, users can type \$ followed by a dot to use auto-completion. They can access visual elements such as screens (\$.Sn), rectangles (\$.Rn), circles (\$.Cn), polygons (\$.Pn), touches (\$.Tn) and measures (\$.Mn). Position and size labels can also be dragged into the code. For example, dragging a rectangle's x-position label into the code generates \$.R1.position.x. Since *design references* refer to their associated visual elements, they reduce reworking and ensure a unique *source of truth* that is always accessible by both designers and developers.

8.2.6.3 *Global and local design references*

Design references can have a global or a local scope. The global scope includes the whole storyboard, while the local scope only includes one particular screen of the storyboard. A *global design reference*, such as `$.R1.position.x`, refers to the x-position of R1 in any screen of the storyboard, i.e. it refers to that design element property at run time. However, developers also need to use particular values from the proposed design, such as an initial color or a particular position or size. This is achieved by using a *local design reference*, which targets a value from a specific screen of the storyboard. By shift-dragging rectangle R1 y-position from screen **S1** to the editor, ENACT generates the *local design reference* `$.S1.R1.position.y`. However, instead of displaying the underlying code as in the global case, a *local design reference* displays its current value, e.g. 237 in this case (Figure 8.5, line 12). However, this value is bound to the visual element in the corresponding storyboard screen. This lets the designer make changes in the storyboard that are automatically reflected in the code, maintaining a single *source of truth*.

Local design references support the story of P12_{dv} in the first study, when he “wrote the code so that the designer could very easily touch it”. In ENACT, designers, as well as developers, can influence the code by directly modifying elements in the storyboard. Other use cases for *local design references* include creating initial/final screens or thresholds such as minimum or maximum values, e.g. to return an element to its original position or to constrain an element’s size to be always smaller than a certain value.

8.2.6.4 *Creating input-output mappings*

In order to program interactive behaviors, developers typically define the rules that connect user inputs with system outputs. To facilitate programming, ENACT provides built-in functions, such as `$.isInside({touch:, shape:})` and `$.map({input:, output:})`. Users can also create their own functions to avoid code duplication and use them on any state machine action.

The `map` function connects changes in input properties, such as the position of a touch, to changes in output properties, such as the position of a graphical object. This alleviates the need for maintaining the initial offset between the touch input and the target shape. For example, dragging rectangle R1 with finger T0 is achieved by `$.map({input:$.T0.position, output:$.R1.position})` in the `touchmove` transition of the default state machine.

The `map` function takes optional additional parameters for further customization: `min` and `max` set the minimum and maximum values of the output property, and `ratio` controls the relationship between the changes in the input and the changes in the output. By default, there is no `min` and `max` value, and the `ratio` is one. More generally, devel-

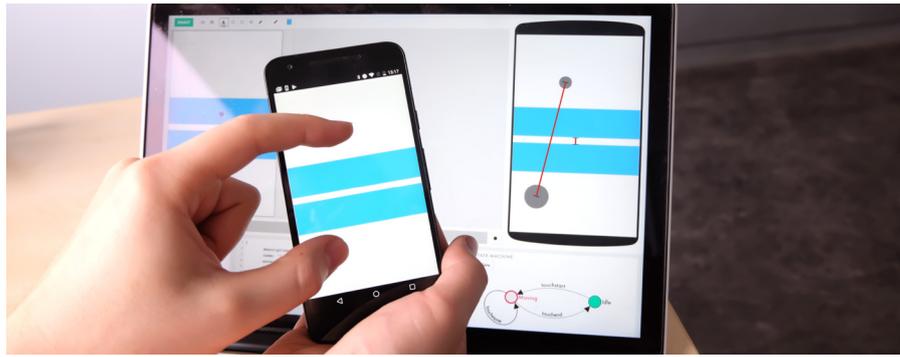


Figure 8.6: The designer created two measures, one between the touches and the other between the two rectangles. These measures are invisible on the target device (left) but they are revealed in the device mirror (right) and updated in real time to facilitate debugging.

opers have access to the full power of JavaScript, including variable declarations, control structures and function definitions. Developers can also create and save functions, to facilitate reuse across projects.

8.2.6.5 *Reifying distances with measures*

Developers are used to working with expressions and formulas calculated from properties of visual elements, such as when the size of a rectangle controls the position of another object. Designers, on the other hand, are more familiar with direct manipulation and visual properties. ENACT makes it possible to *reveal invisible* relationships by reifying (Beaudouin-Lafon and Mackay, 2000) them as *measures*.

Measures are first-class visual objects that can be created between two points of interest of a shape, touch input or other measure (Figure 8.6). When a measure starts and ends on the same point, it represents a point of interest, e.g. the middle of a segment. Measures can be dragged into the code to create *design references*, like other visual elements. For example, a measure can be created between two touches to represent the spread of a pinch gesture, and used as a parameter of the map function to control the size of an object.

By making formulas explicit, measures also reduce code duplication and help create one *source of truth* for the design. Measures can also help identify *edge cases*: instead of having to figure out from the code why a formula evaluates to a given value, the user can look at the device mirror where the measures are visualized in the context of the interface elements and the user inputs. For example, it is easy to see on the device mirror if the correct points are being measured or if the extremities of a segment cross during the interaction.

8.2.7 *Target device and mirror*

In the second and third study we observed the use of external devices as a medium to communicate ideas. Current tools involve the target device typically only to test an implementation. ENACT instead encourages early exploration of interactions during prototyping by giving the target device a more active role as a design medium, supporting *design by enactment*.

By default, ENACT shows the first screen on the target device. This contextualizes the design, letting designers and developers evaluate decisions on the device where they will be used. Designers can use the target device to record touch inputs, to see the generated animation in the early stages of the design, and the actual implementation later on.

ENACT features a device mirror that duplicates the visual state of the target device in real time and augments it with visual information such as the position and sizes of the touch points and the measures (Figure 8.6). The mirror therefore *reveals invisible* but important information to understand the current state of the interaction and for debugging purposes. Such information should not appear on the target device to avoid confusion with the actual interface and additional information. The target device and the mirror therefore also *provide multiple viewpoints*.

Designers and developers are encouraged to *design by enactment* by using the target device to try out the interaction as often as possible. In the process, users sometimes generate an interesting state on the target device, either because it is an *edge case*, or because it complements the storyboard. In such cases, ENACT lets them drag a copy of the mirror and insert it as a new screen in the storyboard.

8.2.8 *Assisted testing*

Assisted testing helps maintain *a single source of truth* among *multiple viewpoints*. The first study showed that one key challenge in designer-developer collaboration is to keep the design and the code synchronized. In software engineering, checking the consistency between a design specification and its implementation involves testing. ENACT supports assisted tests to help designers and developers maintain the code and the design representations synchronized when mismatches occur. Assisted testing executes the code with the recorded input and shows incongruences in the corresponding storyboard screen.

When the user presses the Test button, ENACT sets the target device to match the first screen, then synthesizes input events from the recorded gesture¹ and executes them on the target device, triggering the same code as if they were actual user inputs. ENACT compares

¹ Synthetic events are similar to regular browser events but instead of being triggered by user actions, they are executed programmatically.

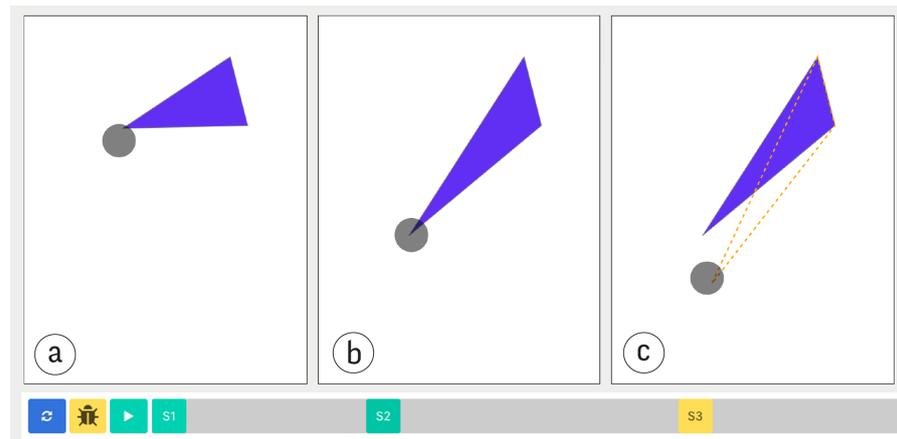


Figure 8.7: Testing in ENACT combines the code with the recorded user inputs. To run the test, the designer presses the Test button (a). Here, the first and second screen match the test result and their screen handles are shown in green (b). The third screen has a mismatch displayed as a dashed orange line and the screen handle is shown in yellow (c).

each screen in the storyboard with the corresponding state of the target device during the test. If the output matches the screen, the corresponding screen handle turns green in the timeline (Figure 8.7a and Figure 8.7b). If there are differences, the screen handle turns yellow and the output is displayed on top of the original design, with the differences highlighted in orange (Figure 8.7c).

After the test, users can drag the screen handle along the timeline and replay the test, as if they were controlling a video of the interaction. This ability to navigate the test results helps designers and developers understand the behavior of the interaction and pinpoint the sources of the differences.

Mismatches between the screens and the test results reveal inconsistencies between the design and its implementation. These inconsistencies break the *single source of truth* principle, requiring manual changes to return the *multiple viewpoints* to a consistent state. Deciding what needs to be modified, either the storyboard, the code, or both, is up to the designer and developer. ENACT helps match the storyboard with the code by snapping the visual objects to the test results when they are moved or resized close to them.

8.2.9 System Implementation

ENACT is a client-server web application developed with Vue, Node.js, Socket.io, CodeMirror and D3². We use reactive data bindings to

² <https://vuejs.org>, <https://nodejs.org>, <https://socket.io>, <https://codemirror.net/>, <https://d3js.org/>

provide liveness within the desktop interface. The target device is connected through an ad-hoc protocol on top of Socket.io messages.

Design references use CodeMirror's text markers to style the code with custom widgets that react to user changes in the storyboard. We extended the CodeMirror parser with regular expressions to support definitions of the form `$. {screen} . {object} . {property} . {sub-property}`. The map function only accepts properties, such as position and size, and sub-properties, such as x-position, y-position, width, and height.

To minimize the learning curve for developers, we wanted to handle properties and sub-properties, i.e. objects and primitive values, uniformly. In particular, these properties and sub-properties needed to keep a record of both their previous and current value. To overcome this challenge, we wrapped these objects behind a JavaScript proxy object that manages *delta values*, i.e. the difference between the current and previous value, and a function that transforms these deltas. Using deltas instead of absolute values simplifies the code by alleviating the need for offsets, such as the distance from the touch position to the shape's origin. The proxy object also adds an interface for the sub-properties so they behave like primitive values, e.g. numbers. Thanks to this approach, from the point of view of the user properties are regular objects and sub-properties are primitive numbers.

8.2.10 *Limitations and Future Work*

We chose to focus ENACT on mobile multi-touch interactions because of the familiarity of the participants with this interaction style and the vast opportunities for customization that it provides. The goal was to illustrate how the design principles informed the design, with the hope that a similar approach can be used for other types of tools. While ENACT supports the creation of advanced multi-touch interactions, it lacks proper support for temporal interactions, such as using timeouts to distinguish between short and long taps, or single and double taps. These could easily be added to the state machine. Also, we focus on continuous interaction, not symbolic gestures that use a recognizer. Such gestures could be implemented by invoking the recognizer at touch-up in the state machine. More generally, applying ENACT to other types of interactions is left for future work.

ENACT uses JavaScript, making it possible to integrate the code into existing development environments. However, manually exporting the code will generate the same problems of reworking and redundancies as with traditional design artifacts. We argue that prototyping tools such as ENACT should be part of an integrated design and development environment. This would support the creation of evolutionary prototypes, a type of prototype "intended to evolve into the final system" (Beaudouin-Lafon and Mackay, 2003). However, if this tight integration is not possible, other lightweight mechanisms such as live

reloading or hot module replacements could maintain *a single source of truth*.

ENACT should support more powerful graphical editing capabilities. Currently, ENACT only provides simple graphical elements but designers are used to the power features of their current tools, such as gradients, compound shapes, text, and imported images. Supporting these features in the context of other *viewpoints* is an interesting area for future work.

ENACT should also support the design of more than one interaction at a time. This will increase the likelihood of conflicts, which could be handled by supporting separate state machines associated with different objects.

ENACT uses a form of programming-by-demonstration but without sophisticated inferencing so as to provide a simpler user interface for professional designers and developers Myers, McDaniel, and Wolber, 2000. However, inferencing could open new possibilities to *maintain a single source of truth*. Currently, when assisted testing detects a mismatch between the storyboard and the code, ENACT highlights the difference. However, ENACT could provide an inference engine that suggests the code modification needed to reach the desired visual state. This type of inferencing could also be bidirectional, i.e. when changes occur in the code, ENACT could suggest the visual modifications needed on the storyboard to maintain consistency.

Finally, ENACT is a single-user application that lets designers and developers work together when they are co-located, or asynchronously when they are remote. In order to facilitate asynchronous communication within the tool, ENACT should support annotations and comments associated with design objects. ENACT should also support versioning system to track changes and to let designers explore and compare alternatives more easily. Finally, designers and developers often want to work concurrently. Supporting real-time remote collaboration introduces extra challenges in order to maintain a consistent, executable design, and is another area for future work.

8.3 STUDY FOUR: ASSESSING ENACT

To better understand how designers and developers interact with ENACT, we conducted a structured observation study (Garcia et al., 2014) with an earlier version of the tool. In terms of the evaluation strategies of Ledo et al. (Ledo et al., 2018), this is a type 2 usage study. We conducted this study with an earlier version of the tool that featured graphical rules to program the interaction (Figure 8.8) instead of the state machine and code editor. The rules provided a visual template for the map function presented earlier. Users could fill out the template by dragging elements from the storyboard. The goal was to enable designers, not just developers, to “program” the interaction.

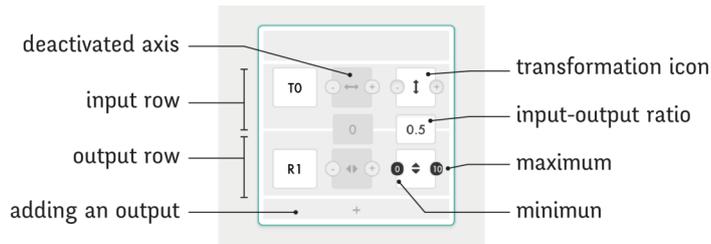


Figure 8.8: Visual specification of rules in the earlier version of ENACT tested in Study Four. Input-output rules connect one input with one or more outputs. The first column shows the identifier of the element (T0 and R1), the second and third columns refer to the X and Y-axis respectively. This rule maps the touch’s Y-axis translation (\updownarrow) to rectangle’s Y-axis scale (\updownarrow) with a ratio of 0.5. Each axis transformation can have a minimum and maximum value taken from the storyboard.

We observed how designers and developers interacted with ENACT and the strategies they used to individually prototype several interactions.

8.3.1 Method

8.3.1.1 Participants

We recruited four participants (1 woman, ages 26-34): two professional developers (P1_{dv} and P2_{dv}) and two professional designers (P3_{ds} and P4_{ds}), who create web sites, mobile applications or interactive installations. Their experience collaborating across disciplines ranges from 3 to 8 years.

8.3.1.2 Apparatus

We run ENACT in a Macbook Pro 13-inch with a 2,7 GHz Intel Core i5 processor and 16GB of memory. The target device connected to the ENACT system was an LG Nexus 5X running Android 7.0.

8.3.1.3 Procedure

We gave a short demonstration of ENACT’s features and asked participants to create three different interactions of increasing difficulty: “simple drag”, “pull down curtain” and “pinch-to-create”. We prompted the first task orally and presented the last two in the form of rough sketches. After the three tasks, we gave participants 15 minutes to experiment freely with ENACT. The study used a think-aloud protocol and took approximately one hour, after which we asked a set of post-hoc questions.

8.3.1.4 *Data Collection*

We recorded audio and video of the participants' interactions with ENACT, on the computer and on the target device. We also took notes during the interviews.

8.3.2 *Results and Discussion*

We performed a thematic analysis (Braun and Clarke, 2006) of the collected data to extract common themes across participants, both during the tasks and the post-hoc interviews. We then revisited the data to specify the themes and to extract relevant quotes.

All participants were able to create the three proposed interactions. Participants were not asked to work quickly, and were encouraged to talk aloud while doing the tasks. Even so, all participants finished the first task in less than three minutes without prior training. All participants finished task 2 in less than five minutes and task 3 in less than 15 minutes.

8.3.2.1 *An embodied perception of interaction*

Without rules relating touch inputs and shapes, a prototype is not interactive. We were surprised to see all participants try to interact with the shapes on the mobile device before they create a rule. Both designers and developers wanted to immediately explore interactivity during prototyping. P_{1_{dv}} noted that ENACT approaches interaction from "*a sensible point of view, just like the end user would experience it on the mobile*".

All participants interacted with the target device as soon as they created rules. Thanks to the live nature of the system, P_{1_{dv}} realized that one of his rules was incomplete: "*Now I realize that it grows only downwards, I need to move it up.*"

Participants also liked the assisted testing. For example, after the first interaction, P_{2_{dv}} decided to rely exclusively on the assisted testing to verify the rules. As P_{4_{dv}} could not understand why the interaction on the target device did not react as expected, he ran the assisted test and slowly navigated the history of the test results to find the problem.

8.3.2.2 *A pedagogical tool*

Both designers and developers saw the value of the tool to help them collaborate across disciplines. For example, P_{3_{ds}} explained: "*Today, I try to create simple things so that the developer does not have to get headaches while implementing them*". P_{3_{ds}} added: "*What is interesting is the pedagogical aspect. I am forced to ask myself what I'm gonna ask the developer.*"

On the developers' side, P_{1_{dv}} thought that this tool would help designers better understand custom interactions: *"There are not too many constraints at the beginning. It shows that it can be fun and quick to create things that are very hard to implement today. But you can also increase the complexity afterwards"*.

8.3.2.3 *The need for more power*

Participants were able to use ENACT's features but, as they engaged with the tool, they wanted to manipulate finer details of the interaction. For example, P_{1_{dv}} wanted more control over the rules: *"I want to add this delta value. I want operators to correct rules."* P_{2_{dv}} also wanted to be able to link shapes and values in the rules, so that he could directly modify the latter by moving the corresponding shapes in the storyboard. These remarks led us to replace the visual rules with the map function and introduce *design references* in the current version of ENACT.

8.4 STUDY FIVE: COMPARING ENACT WITH TRADITIONAL TOOLS

Based on the feedback from the previous study, we created the version of ENACT described in [Section 8.2](#). We replaced the graphical rules with the interactive state machine diagram and added the augmented code editor to provide more control.

We then conducted a second study with this new version in order to better understand how ENACT affects designer-developer collaboration. The goal was to observe and compare the strategies used by designer-developer pairs to represent, communicate and implement interactions with their own tools and with ENACT. We focused on the issues that arise from the inability to successfully represent the interaction itself, rather than those that arise from generating design ideas. We therefore used the same strategy as in Study Three and provided interaction examples from existing applications that participants had to describe and implement. For ecological validity, we organized the study in three phases that reflect the common collaboration patterns we observed involving a design hand-off: communication of the initial design (designer only), initial implementation (developer only), and side-by-side collaboration (designer and developer, co-located).

8.4.1 *Method*

8.4.1.1 *Participants*

We recruited 12 participants (6 women and 6 men, ages 23-35): six professional developers (P_{1_{dv}} to P_{6_{dv}}) paired with six professional designers (P_{1_{ds}} to P_{6_{ds}}), who create web sites, mobile applications or interactive installations. Their experience in collaborating across

Condition	Traditional tools					Enact	
	Initial version		Final Version			Initial version	Final Version
P_{ds}	10 min		15 min	Enact intro and training	10 min		15 min
P_{dv}		15 min				15 min	

Figure 8.9: Study Five. Each pair consist of one designer (P_{ds}) and one developer (P_{dv}). The designer has 10 minutes to create a design artifact for the initial version of the proposed interaction, then the developer has 15 minutes to implement it. Finally, both work together for 15 minutes to evaluate the current implementation and work on the final version of the interaction.

disciplines ranges from 0 to 7 years. P_{1dv} and P_{6dv} reported no significant collaboration experience as they were just starting their front-end developer careers.

8.4.1.2 Apparatus

Participants used their own setup, i.e. laptop, mouse, and pre-installed software, to work with their TRADITIONAL tools. We provided two LG Nexus 5X running Android 8.0 and an Apple iPhone 6S running iOS 10 as the target mobile devices. We helped participants to connect these devices to their development environments before starting the study. We run ENACT in a Macbook Pro 13-inch with a 2,7 GHz Intel Core i5 processor and 16GB of memory. The target device connected to the ENACT system was the LG Nexus 5X. The other LG Nexus 5X was used to show examples of interactions to the designer.

8.4.1.3 Procedure

The study uses a think-aloud protocol and takes approximately two hours. After each condition we ask participants to fill out a short questionnaire. At the end of the session, we conduct a post-hoc interview.

Each pair first creates an interactive prototype of an existing interaction with their preferred tools (TRADITIONAL condition), then does the same for a different interaction with ENACT (ENACT condition). For the TRADITIONAL condition we do not impose any restriction on the tools they can use. We only let them know that they will work for a mobile platform supporting multi-touch interactions. To avoid any influence from ENACT on the participants' usual workflow, they all perform the TRADITIONAL condition before ENACT. Once the TRADITIONAL condition is over, we give a short presentation of ENACT and let each participant individually practice for 10 minutes before performing the ENACT condition.

For both conditions we follow the same protocol (Figure 8.9). First, we show the designer an initial version of an interaction preloaded on one of the mobile devices. Only the designer has access to this device throughout the study. The designer has 10 minutes to create a design that communicates all the details he deems relevant for creating a prototype with the same behavior. Then the designer sends the document to the developer, who has 15 minutes to create an interactive prototype based on the received design. During this time, we show a final version of the interaction to the designer. Finally, we give the pair another 15 minutes to review the initial implementation and work together to prototype the final version of the interaction.

For each condition we use a different multi-touch continuous interaction with two versions: initial and final. This lets us study the introduction of changes in a controlled environment: the initial version features the basic interactive functionality while the final version includes changes and adds complexity to the previous design. We use two custom interactions already used in the participatory design workshop (Study Three, Section 7.3): *pinch-to-create* and *pan-and-stamp*. We simplified certain aspects of the visual design such as gradients, texts and 3D effects to focus on interactivity. Both interactions feature at least seven *edge cases*. We expect these interactions to have a similar level of difficulty.

The first interaction, *pinch-to-create*, is inspired by the *Clear* to-do list mobile app³, which uses a spread gesture to create a new item between two existing items (Figure 7.5). The initial version lets users simultaneously move two rectangles with a pinch gesture. The final version lets users manipulate the size of a third rectangle with the spread of their fingers. The *edge cases* include: The interaction should only work when the fingers are inside the rectangles; The third rectangle is always positioned in the middle of the other two; The upper rectangle cannot move lower than its starting position.

The second interaction, *pan-and-stamp*, is inspired by the *Paper* note-taking mobile application's⁴ cut and paste feature, where one finger drags the object and a tap with another finger copies it. The initial version lets users pan a rectangle with one finger and copy the rectangle with another finger tap. The final version lets users pinch to resize the rectangle and, with a third finger, tap to create new copies of it. The *edge cases* include: New rectangles are centered at the tap position; The first rectangle needs to be resized from the center, not the top-left corner; and should also be panned with two fingers. The two interactions are balanced across pairs: P1, P2 and P4 start with *pinch-to-create* while P3, P5 and P6 start with *pan-and-stamp*.

³ <https://www.realmacsoftware.com/clear/>

⁴ <https://www.fiftythree.com/paper>



Figure 8.10: P_{4ds} and P_{4dv} working side-by-side on the final version of *pan-and-stamp*. On the left, the developer performs an off-device mimicking gesture with his left hand to understand the proposed design. On the right, the designer performs an on-device mimicking gesture with both hands to communicate the design.

8.4.1.4 Data Collection

We recorded audio and video, both over the shoulder and with a screen recorder. We also took notes during the tasks and the post-hoc interviews.

8.4.2 Results and Discussion

We performed a thematic analysis (Braun and Clarke, 2006) of the collected data to extract the prototyping strategies used by the participants. After looking for themes, we revisited the data to specify the classification and extract relevant quotes. We also measured the number of mimicking gestures in the video data, i.e. when participants acted out the interaction with their hands, either to understand it or to communicate it, both away from the device and on the device. In our initial studies we observed designers and developers using hand gestures to communicate the interaction. We were therefore interested in whether ENACT encourages participants to use more mimicking gestures when collaborating.

We focused our analysis on the side-by-side collaborative phase (not the individual phases) and on on-device mimicking gestures (Figure 8.10). We also measured the amount of edge cases found by each pair from the notes and the video data.

8.4.2.1 Tools of choice

TRADITIONAL CONDITION The six designers chose vector and raster graphics software: Sketch (5/6) and Photoshop (1/6). The six developers chose mobile web (3/6) or mobile native (3/6). The developer web tools included browsers, such as Google Chrome (2/3)

and Mozilla Firefox (1/3), and code editors, such as Sublime Text (2/3) and Atom (1/3). The developer native tools were IDEs: Android Studio (2/3) and Apple Xcode (1/3). All developers used search engines to look for terms such as “JavaScript pinch” or “Android button onpress”, complemented with inline help when available, such as Android Studio and Xcode documentation. Pair P₁ never used the provided mobile device and only used the on-screen emulator. Pairs P₂ and P₃ used the mobile device but relied almost exclusively on the on-screen debugger of the web browser, even though it does not support multi-touch input.

ENACT CONDITION Participants did not chose a tool on this condition. We provided a laptop with ENACT pre-installed and connected to a mobile device.

8.4.2.2 Task completion

TRADITIONAL CONDITION No pair completed the final version and only one pair (P₄) finished the initial version of the provided interactions. P₄ was also the only pair that started the implementation of the final version of the interaction in the TRADITIONAL condition. Two of the three pairs that started with *pinch-to-create* implemented its basic functionality, but none of the pairs that started with *pan-and-stamp* finished it.

ENACT CONDITION All pairs provided the basic interactivity of the initial version of the interaction. Five out of six pairs provided the basic interactivity of the final version and one of them implemented all the details of the final interaction. All pairs implemented the basic and extended versions of *pinch-to-create* and the basic interactivity of *pan-and-stamp*. Two of the three pairs also implemented the extended version of *pan-and-stamp*. Despite our efforts to use interactions with similar complexity, we found that *pan-and-stamp* seemed more difficult to prototype than *pinch-to-create*. However, this should not pose a threat to validity because these techniques were counterbalanced across pairs.

Since ENACT is explicitly designed to prototype the type of interactions provided to the participants, we expected a higher completion rate with ENACT than with TRADITIONAL tools. On the other hand, no participant was familiar with ENACT and they only had a short training session. Even so, participants were able to complete much more with ENACT than with their TRADITIONAL tools. Since we gave participants very little time and did not expect them to finish all the tasks, we concentrate the following points on the their strategies to prototype the interaction rather than on performance measures.

8.4.2.3 *Transitioning problems*

TRADITIONAL CONDITION All the designers (6/6) used the same workflow to communicate the design to the developer in the TRADITIONAL condition. Designers created a storyboard document depicting the visual states of the interface with a graphical software. Then, they sent it to the developer either in the original format, as a PDF, or through a specialized tool such as InVision. Designers illustrated the visual design with different screens and explained the user inputs with circles, icons, traces, text annotations or a combination of these. None of the designers used animations nor video to communicate interactivity. P6_{ds} mentioned her intention of using video but she felt that it would require too much time with her TRADITIONAL tools. None of the designers updated their design specification at the beginning of the side-by-side phase to communicate the new design. Rather than using a design artifact, they described the final version of the interaction on top of the current implementation, verbally or by using mimicking gestures.

In the TRADITIONAL condition, developers ran into problems when interpreting the interactivity and reproducing the visual look. Developers expected text annotations. For example, P5_{dv} said “*I don’t understand this*” when viewing the design for the first time. When text annotations were minimal, developers expected more details. For example, P3_{dv} said “*I don’t know if these are multiple interactions or different steps of the same interaction*” and that she “*prefer[s] comments saying ‘when this happens then that happens’*”. Four developers ignored the graphic design and used either no visual elements at all (only console logs), gray buttons or wrongly colored rectangles. Two developers used external color pickers to extract the right color from the design and copy the corresponding hexadecimal string. All the developers ignored the precise size of the rectangles: for *pinch-to-create* the height of the rectangle in the design was not replicated and for *pan-to-stamp* developers generally used rectangles instead of the square shown in the design.

ENACT CONDITION Designers used the animation feature to refine the storyboard and developers used it to understand the interaction. Thus, the generated animation worked as a contact point between the two activities while working asynchronously. Only one designer asked for icons to represent user input and another expressed the need for text annotations. ENACT does not currently support text annotations but developers did not mention the lack of this feature. One explanation for this could be the use of ENACT’s generated animation as a communication medium instead of textual descriptions. Developers also mentioned the usefulness of showing the touch input information on the device mirror while the animation was being played.

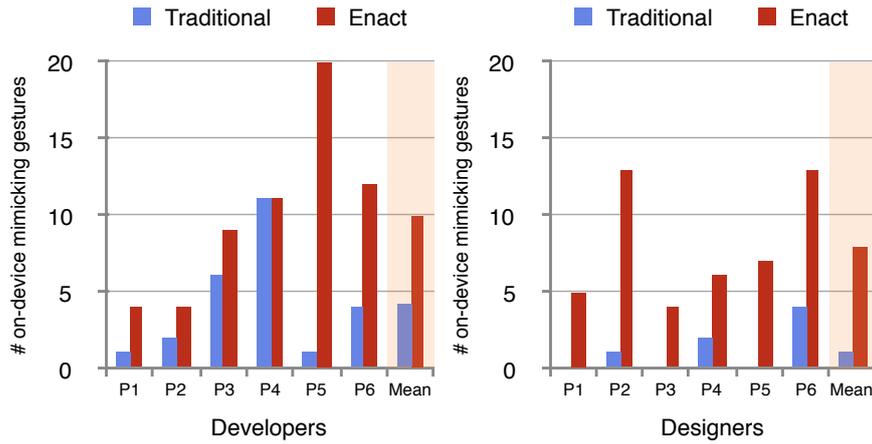


Figure 8.11: Number of on-device mimicking gestures, either on the mobile device or the emulator provided by the IDE, per participant during the collaborative side-by-side phase. Designers and developers performed significantly more interactions on the device with ENACT than with TRADITIONAL tools. This suggests that with ENACT, designers participate more during the side-by-side phase and developers perform more contextualized actions on the real device during implementation.

In the TRADITIONAL condition, all developers used print logs to confirm the triggering of input events. ENACT’s live state machine diagram provided the same level of confidence to the developer, without extra effort and with more detail. In the TRADITIONAL condition, $P4_{dv}$ finished the basic interactivity of *pinch-to-create* but had issues to preserve the right offset between the touch point and the shape during the interaction. In the ENACT condition, while using the *map* function to implement *pan-to-stamp*, he emphatically expressed that “*it is really cool that I don’t have to think about the sh*tty offset*”.

The reduction in rework and redundancies was evident throughout the study. For example, four out of six developers did not follow correctly the user interface specification with the TRADITIONAL tools. In ENACT there is no transition step, eliminating these issues altogether. With TRADITIONAL tools all the designers relied on copy-paste instead of using “smart objects” to create the design specification. In ENACT, the storyboard propagation helps designers maintain consistency between the storyboard visual elements. Moreover, none of the designers used a design artifact to communicate the second task with their traditional tools. Instead, they relied on mimicking gestures or verbal communication. With ENACT, the design artifacts are part of the prototype and were heavily used in both tasks.

8.4.2.4 Increasing participation

Both designers and developers performed significantly more interactions on the device with ENACT than with TRADITIONAL tools ($F=1.4$;

$p < .007$) (Figure 8.11). This suggests that designers were much more involved during the side-by-side collaboration with ENACT than with TRADITIONAL tools. With ENACT, all designers interacted with the target device ($M = 8$ times, $SD = 3.65$) while only three designers did it with TRADITIONAL tools ($M = 1.17$ times, $SD = 1.46$).

One reason for the increased participation is that developers were faster in creating a functional implementation with ENACT, which gave designers and developers time to collaborate on the created artifact. Also, even if the implementation was not finished, with ENACT, the designer had “something to play with” while with TRADITIONAL tools they usually did not. Nevertheless, we found instances, in both conditions, where designers and developers were acting out the interaction on the device even when the implementation was not finished.

Another explanation for this increased interaction with the target device could be the sense of *ownership* of the prototype. With TRADITIONAL tools, developers recreate the design with their own tools. It is not the designer’s design that “comes to life” but a mere replica. With ENACT, developers literally add interactivity to the artifact provided by the designer. Designers might therefore feel a stronger sense of ownership over the prototype under construction, thus increasing participation. P_{4ds} said that “*you have the impression to be living in the same environment, that we share the same language*”.

8.4.2.5 Finding more edge cases

TRADITIONAL CONDITION Only two pairs (2/6) found one and two of the seven edge cases –remember that only one pair finished the basic and final version of the interaction in this condition. For example, while coding *pinch-to-create* in a text editor, P_{2dv} did not notice that the to-do list items should only move along the y-axis. He noticed this edge case only when he tried the interaction in the simulator. However, the simulator runs on the desktop browser instead of the target device, so that the mouse cursor events are treated as touches of a single finger, making it impossible to find edge cases related to the use of multiple fingers. In contrast, P_{4ds} found an extra edge case because he was interacting with an actual multitouch device: The Apple Xcode IDE was connected to a phone, and he noticed that the items moved beyond their initial position. However, P_{4dv} needed to investigate the implementation carefully in order to find which piece of code was in charge of the erroneous behavior. The lack of a clear connection between the runtime effect and the build-time code hindered P_{4dv}’s ability to quickly fix the issue.

ENACT CONDITION Five pairs (5/6) found three or more of the seven edge cases in the final version of the interaction. For example, while interacting with her first implementation attempt, P_{3dv} realized that the shapes were dragged even when her finger was outside of

them. Similarly, P_{5ds} noticed that *pinch-to-create* behaved differently depending on which rectangle was touched first. He shared his finding with P_{5dv}, who added an *if* statement to determine which touch should be associated with each rectangle.

ENACT interactive representations helped designers and developers find more *edge cases*. ENACT contextualizes the interaction in the final device, giving developers and designers insights that are difficult to find in their indirect TRADITIONAL tools: Emulators do not provide the same experience that the final context of use; IDEs connected with a real device impose a time-consuming build-run-test cycle; Graphic-authoring tools are focused only in the visual properties of the interaction. ENACT provides a faster design-test cycle, with more chances to find edge cases on the design by actually interacting with it. In other words, it is easier to find edge cases because the whole interaction lives less on the participants' head and more on their hands.

8.4.2.6 Opportunities for co-creation

TRADITIONAL CONDITION Designers took a more passive role and were often just observing the developer. In the two occasions where they communicated, the participation of the designer was minimal. P_{2dv} had an emulator on his screen and P_{2ds} had to stand up every time she wanted to point to the screen to make a comment. In contrast, the target device was positioned between P_{4dv} and P_{4ds}, facilitating their collaboration and participation. Nevertheless, during the side-by-side phase only the developer modified the interaction, while the designer only made suggestions.

ENACT CONDITION ENACT creates a kind of "gray area" between design and development with interesting opportunities to cross boundaries. P_{2ds} was not sure she should create an input example: "*am I supposed to do this?*". On the contrary, P_{5ds} was really interested in the programming capabilities of ENACT: he started adding interactivity by himself and forgot to finish the description of the interaction on the storyboard. When the developer of the same pair (P_{5dv}) received the design, he said "*What should I do now? This is already coded!*".

We observed that providing *multiple viewpoints* helped break the silos between the two communities. With ENACT, most of the designers were intimidated by the code editor but not by the live state machine diagram. During the side-by-side collaboration, P_{5dv} built on top of P_{5ds}'s implementation, even extending the storyboard himself. Similarly, P_{6ds} added some interactivity to the *pinch-to-create* prototype and P_{6dv} directly started to fix several *edge cases* in the implementation, such as checking that the touches are inside the shapes and that the

rectangle is constrained to a vertical movement. Finally we observed that intermediary representations, such as state machines, align the designer-developer vocabulary for abstract concepts, such as transition events or the state of the interaction.

8.4.2.7 Summary

We observed that ENACT reduced rework and redundancies, allowed designers and developers participate more, helped them find more edge cases and provided a friendlier environment to co-create with the other community of practice. Both designers and developers performed significantly more mimicking gestures on the device with ENACT than with TRADITIONAL tools, indicating a higher degree of participation. With ENACT, five out of six pairs found three or more of the seven edge cases in the final version of the interaction. With the TRADITIONAL tools, only two out of six pairs found one or two edge cases. Finally, ENACT helped designers and developers to cross their boundaries, creating more opportunities to co-create the interaction. We observed that providing *multiple viewpoints* while maintaining *a single source of truth* helped break the silos that traditionally insulate the two communities of practice. Designers were able to manipulate developer-oriented artifacts, such as the state machine and the code editor, thanks to the use of *design references*. On the other hand, developers also manipulated the storyboard and the target device, either to modify the recorded inputs, or to reveal touch and measure information on the device mirror.

8.5 CONCLUSION

Despite their different backgrounds and skills, designers and developers need to collaborate to create interactive systems. Our goal was to investigate how better prototyping tools can support their collaborative process.

This work makes three main contributions:

- At the empirical level, we report on three studies to better understand and classify current collaboration issues;
- At the theoretical level, we introduce a set of principles to design better collaborative prototyping tools; and
- At the technical level, we present a new tool guided by these principles to reduce collaborative breakdowns during the creation of custom touch-based interactions.

In study one we showed that current workflows and tools induce unnecessary rework. We found that designers create a multitude of redundant design documents and developers must recreate them with

their own tools. This process often introduces mismatches with the original design. We proposed a classification of key design breakdowns: *missing information*, when designers do not communicate a specific detail; *edge cases*, when designers do not think about a particular case; and *technical constraints*, when designers are not aware of developer's technical limitations. The interviews also showed that when developers are not involved in the initial design phase, implementation tends to be problematic or even impossible.

In study two we found that even if the early involvement of the developer mitigated *design breakdowns*, new breakdowns appeared in subsequent meetings. We also found that the inability of current design artifacts to represent interaction generates breakdowns.

In study three we observed that the limitations on the representations used to communicate interaction result in *missing information* and ignored *edge cases*. The successful communication and representation of interactions required an iterative process, from concrete examples to more general rules.

Based on this empirical work, we introduce four principles for collaborative prototyping tools to reduce designer-developer breakdowns: *Provide multiple viewpoints*, to allow developers to participate early, *maintain a single source of truth*, to reduce reworking and redundancies, *reveal the invisible*, to avoid *missing information*, and *support design by enaction*, to find more *edge cases*. Collaborative prototyping tools based on these principles can play the role of a *boundary infrastructure* (Bowker and Star, 2000) that supports the flow of multiple interconnected objects between designers and developers.

To demonstrate these principles in action, we created ENACT: a novel interaction prototyping tool for touch-based interaction on mobile devices. Through multiple interconnected representations of the interaction under construction, ENACT reduces reworking, redundancies and design breakdowns. Storyboard propagation reduces redundancies within representations while *design references* reduce redundancies across representations. The interactive state machine diagram and the device mirror let designers and developers quickly explore the interaction under construction and detect *edge cases*. The connected target device lets designers and developers enact the interaction at each stage of the process.

Finally, we conducted two studies of ENACT to validate our design goals, gather feedback from professional designers and developers, and analyze the impact of ENACT during collaborative prototyping. In the first study, participants adopted *design by enaction* and appreciated the pedagogy of the tool to understand the details of the interaction. Both groups highly appreciated the reduced time-to-interaction, but wanted more powerful tools to describe the relationships between user inputs and system outputs. Based on this feedback, we added two

new *viewpoints* for finer control: an interactive state machine and a code editor with *design references*.

In the second study, we assessed the new version of ENACT in a collaborative setting and compared it with traditional tools. We found that participants completed more of the proposed interaction with ENACT than with traditional tools. All the pairs finished the initial task while only one pair finished it with the traditional tools. Participants not only interacted more but they also found more *edge cases* in the design. We also found that with ENACT, both designers and developers performed significantly more mimicking gestures on the device than with traditional tools, and designers were much more involved during the side-by-side collaboration. This shows that ENACT encourages a more active role during design and development, compared with the passive or indirect approach encouraged by traditional tools.

Our future work will focus on improving ENACT, studying designer-developer collaborative prototyping in the wild, and going beyond touch-based interactions. Evaluating ENACT in the wild will let us study the effects of collaborative prototyping on a long-term project, and hopefully will provide interesting findings about the use of interactive prototypes to collaborate with other stakeholders, such as clients, users, testers, and managers. We would also like to apply the design principles to other interaction styles, such as mid-air gestures, multimodal interaction or mixed reality.

FROM THROWAWAY PROTOTYPES TO TAKEAWAY PROTOTYPING

Previous chapters analyzed video prototypes and interactive prototypes in isolation. This chapter discusses the underlying theoretical principles that grounded the design of the presented tools. It also outlines a new prototyping approach called Takeaway Prototyping.

9.1 REIFICATION, POLYMORPHISM, AND REUSE

The principles of reification, polymorphism, and reuse introduced by Beaudouin-Lafon and Mackay (2000) strongly influenced this dissertation. Beaudouin-Lafon and Mackay (2000) presented these three principles to guide the creation of powerful interactive systems. I will present the principles in the context of interaction design with a particular focus on how they were applied in the previously presented prototyping tools.

9.1.1 *Reification*

To reify is to transform a concept into an object. While this idea emerged in antique philosophy, it is also a common practice in programming. In the context of interaction design, to reify is to transform a command into an object which the user can manipulate (Beaudouin-Lafon and Mackay, 2000). Reification is a key concept in Instrumental Interaction (Beaudouin-Lafon, 2000). Instrumental Interaction is an interaction model that extends Direct Manipulation (Shneiderman, 1983). It describes, compares and generates user interfaces in terms of reified instruments, i.e. mediators between the user actions and the object of interest (Beaudouin-Lafon, 2000). For example, a [WIMP](#) scrollbar can be seen as an instrument that indirectly controls the positioning of the document –the object of interest– through manipulations over a mouse input device –the user actions–. An instrument has a logical and a physical part, e.g., the logical instrument is the scrollbar widget on the screen, while the physical part is the mouse input device. Some examples from the tools I have presented in this dissertation:

IN VIDEOCLIPPER I reified the alignment action into the ghost image of the last recorded frame; Designers can toggle it on or off and also change its opacity.

IN MONTAGE I reified the action of selecting an area of the paper prototype into the viewport tool; Designers can sketch the area of interest, pan to change its position or pinch to change its size.

IN ENACT I reified the action of calculating the distance between two objects into the measurement tool; Designers can materialize the desired distance in the storyboard by just connecting the two objects of interest. Afterward, the measure is revealed on the device's mirror and it is available on the code editor without the need to program an equation.

9.1.2 *Polymorphism*

Creating polymorphic entities is a well-known programming technique. Two or more objects are polymorphic if they have different types (Cardelli and Wegner, 1985) and another object can use them indistinctly. Polymorphism and reification are intimately related. With reification, the number of objects in the system increases as well as the complexity of manipulating them. Polymorphic objects share a common communication protocol in order to reduce this complexity.

In the context of interaction design, polymorphism "is the property that enables a single command to be applicable to objects of different types" (Beaudouin-Lafon and Mackay, 2000). Some examples from the tools I have presented:

IN VIDEOCLIPPER the moving command is polymorphic for the TitleCards, the video clips, and the Lines i.e. they can all be dragged with the same gesture in the interactive storyboard.

IN MONTAGE the command to open the color palette, i.e. long press, works polymorphically over the camera preview and the individual sketches. The former action sets the global color, while the latter only changes the color of the selected sketch.

IN ENACT the action of creating a *design reference*, i.e. control-dragging an object from the storyboard to the code editor, works over shapes, measures and touches indistinctly.

9.1.3 *Reuse*

Good developers avoid code duplication. Methods and techniques to avoid duplication started to be formalized in the 1960s and 1970s with the introduction of structured programming (Dahl, Dijkstra, and Hoare, 1972). For example, the Don't Repeat Yourself (DRY) principle states that "every piece of knowledge must have a single, unambiguous, authoritative representation within a system" (Hunt and Thomas,

1999). If code duplication were a disease then code reuse would be the cure.

In the context of interaction design, reuse "can involve previous input, previous output or both" (Beaudouin-Lafon and Mackay, 2000). With input reuse, previous users inputs are recycled, e.g., the *redo* command repeats past actions without the need to perform the actions again. With output reuse, previous system outputs are recycled, e.g., the *copy-paste* commands replicate existing text or images without the need to recreate them manually. Some examples from the tools I have presented:

IN VIDEOCLIPPER the ghost image is an example of output reuse. It reuses the last recorded frame as an alignment guide for creating rough invisible cuts.

IN MONTAGE the creation of animation by demonstration is an example of input reuse. It recycles user inputs over the sketches during the live *WOz*, such as moving, scaling and rotating, to create keyframe animations synchronized with the underlying video clip. The green screen replacement allows output reuse, i.e. the same context can be used multiple times to illustrate different interface videos.

IN ENACT *supporting design by enaction* leads to input reuse. For example, when the designer act outs the desired gesture, the recorded input events are reused, e.g., they represent touches on the storyboard and they guide the execution of the assisted testing.

9.2 INFORMATION SUBSTRATES

The previous three principles, i.e. reification, polymorphism, and reuse, describe desired properties of the interactive objects in the system under construction. However, what is the underlying structure that these objects need to have to exhibit these properties? Information Substrates is an on-going theoretical effort to answer that precise question:

"A substrate is a digital computational medium that holds digital information, possibly created by another substrate, applies constraints and transformations to it, reacts to changes in both the information and the substrate, and generates information consumable by other substrates. Substrates are extensible, composable with other substrates, and they can be shared. They provide the fabric of the digital world" Beaudouin-Lafon (2017).

The concept started to be explored in music authoring tools (Garcia et al., 2012), web environments (Klokmoose et al., 2015) and graphic

design (Maudet et al., 2017a). In this dissertation, I explored multiple information substrates focused on the structured representation of interactive behaviors. I found various types of information substrates that I describe as non-visual, visual and interactive. A non-visual information substrate represents a structure inaccessible to the user but that enables objects to support the three principles. A visual information substrate represents a structure that supports the principles and can be seen by the user. An interactive information substrate is a visual substrate that is also manipulable by the user.

In ENACT, the main information substrate is the non-visual structure that allows the integration of multiple viewpoints. For example, the substrate presents measurements as visual entities in the storyboard as well as symbolic *design references* in the code. This is achieved in the ENACT's internal programming model by having two different view objects and making them share the same model object—similar to the approach found in the Model-view-controller (MVC) architectural pattern (Krasner and Pope, 1988).

This substrate enables the reification, polymorphism, and reuse of the multiple representations of the measure. The measure *reifies* the distance between any two objects, such as touches, shapes, and even other measures. The measure also acts *polymorphically* over objects of different types, i.e. a touch and a shape, a measure and a touch, or a measure and a shape. Finally, designers *reuse* measures as a debugging tool, e.g., to assess the gesture behavior in the device mirror, and as a programming construct, e.g., to assign it as the input of a mapping function. However, this substrate is not available to the designers, they only interact with the objects built on top of this hidden structure, not the structure itself.

In MONTAGE, the green area in the context videos, i.e. what the UserCam records, is an example of a visual substrate. The green area allows context and interface videos to be combined. The same context video can be composed polymorphically with interface videos or digital sketches. This substrate is not hidden, however, even if designers can see the substrate they cannot directly manipulate it; it is MONTAGE—and not the designer—the one that internally interacts with this structure during composition.

In VIDEOCLIPPER, the interactive storyboard is an interactive substrate. Its two-dimensional visual structure organizes TitleCards and videos in a grid of objects. Lines reify the grouping of TitleCards and videos to form meaningful narratives. Designers can manipulate these Lines or the individual elements polymorphically. Also, designers can clone Lines to reuse previously created scenes.

Video file containers, such as MPEG4 and QuickTime, serve as time-based information substrates. These file containers treat video as a collection of time-based tracks. The typical tracks of this structure are audio and video, but other tracks such as subtitles, event logs

or additional video tracks, e.g., filmed from other angles, can be embedded in the same file. This substrate is non-visual, e.g., when the user opens a video prototype file in a regular video player. However, this substrate can be interactive, e.g., when the same file is opened on a video editing software so that designers can see, replace or edit the individual tracks.

All the presented tools used time-based information substrates. Both VIDEOCLIPPER and MONTAGE create video prototypes supported by a time-based multimedia file. The interaction sequence is fixed; designers can navigate this sequence by manipulating the time forward or backward. Despite the limited interactivity of video –they are only watchable–, time enables a simple vocabulary to explore examples within a design team. For example, instead of referring to an interaction as "when the user picks up the phone, opens the menu and selects the drawing tool" designers can simply fast-forward to the desired point in the movie file.

ENACT also uses time-based information substrates as a structure that connects multiple instruments or objects. The timeline saves the user inputs as they were demonstrated by the designer. Each screen of the storyboard is associated with one of these timed user inputs. In order to create an automatic illustration, ENACT creates a keyframe animation. Each keyframe needs an image and a timestamp: each storyboard screen is the key image and the duration of the interpolation is the time between each associated user input.

9.3 TAKEAWAY PROTOTYPING

Besides having the same theoretical foundation, the presented tools support a similar style of prototyping that I call Takeaway Prototyping.

In all the tools the designers need to put themselves in the shoes of the user¹. This idea is similar to Experience Prototyping:

"an Experience Prototype is any kind of representation, in any medium, that is designed to understand, explore or communicate what it might be like to engage with the product, space or system we are designing" (Buchenau and Suri, 2000).

However, Experience Prototyping is presented only as an "attitude" towards prototyping without any tool support (Buchenau and Suri, 2000).

Takeaway Prototyping extends Experience Prototyping towards a prototyping approach supported by digital tools. In contrast to throw-away prototypes that emphasize the disposability of the artifacts, the

¹ In a Participatory Design session the same instruments used by designers can be used by actual users but we will call the user of the tools designers as in the previous chapters

presented tools emphasize the most important aspect of the prototypes: the takeaways, i.e. the things that we learn from them. In Takeaway Prototyping the designers should be able to:

1. Design by enaction; and
2. Reuse design artifacts.

9.3.1 *Design by enaction*

As introduced in [Section 8.1.5](#), design by enaction proposes that prototyping tools should allow designers to describe interaction with their own body enactments such as hand gestures, voice commands or gaze. Designers should start designing by enacting the interactions of the system under design. In ENACT, the designer performs the gestures on the actual device where the final interaction will be executed. This lets designers feel the interaction during this early prototyping stage. While designers can start from any of representation, e.g., the code editor, we encourage designers to start from the more concrete representations, such as the storyboard or the recording of the gesture. However, ENACT is not the only tool that *supports design by enaction*.

Both VIDEOCLIPPER and MONTAGE rely on contextual demonstrations of the interaction with the system under design. Apart from acting the user inputs, the user-actor plays a character in the story. Contextualizing the use of a system in a story helps designers think on concrete use scenarios, which pushes them to reflect on detailed user requirements and technology breakdowns, e.g., what is the overall motivation of the user? what is the user trying to achieve? did it accomplish it?

I presented concrete prototyping tools that use the designers demonstrations as inputs to sketch representations of the interaction. This idea is similar to programming by example (Myers, McDaniel, and Wolber, 2000) but instead of finding or inferring a particular and pre-defined system behavior, the examples –or enactments– are used to explore and experiment with the design material.

9.3.2 *Reuse design artifacts*

These tools not only support *design by enaction* but they reuse the generated enactments throughout the prototyping process. The demonstration communicates interactive ideas, e.g., the video prototypes generated by VIDEOCLIPPER or MONTAGE, or the demonstration itself evolves into a more general interaction behavior, e.g., in ENACT certain elements of the demonstration –such as the touch identifiers– are part of a map in the code editor. These demonstrations should have a "high degree of compatibility" with the final context of use, e.g., when designers are prototyping an interaction that requires the user to walk

outside, the designers should be able to give a demonstration in the same context. This also applies to the physical mediators between user and computer interaction. The system under design, either a mobile device, a wearable, a wall-size display, a robot arm, etc., needs to be part of the design elements, not only as an environment to test the implementation but as a design tool.

Instead of only focusing on the prototyping speed and the prototype's disposability, I think prototyping tools should be able to materialize the lessons learned among iterations. Today, iterating over a throwaway prototype requires designers to persist the lessons learned only in their minds and replicate them in a new iteration starting from a blank canvas. Prototyping tools should let designers persist these ideas in the tool itself in order to work on top of them. For example, in ENACT, the assisted test persists the expected design and the current computation to maintain both activities in sync.

Takeaway Prototyping differs from iterative prototyping in two main ways. First, Takeaway Prototyping aims to be closer to the prototyping speed of rapid techniques by encouraging the use of *design by enactment* instead of the use of symbolic representations. Iterative prototyping techniques provide a higher level of precision than rapid prototypes. However, they require the use of either visual or textual programming languages decreasing the prototyping speed of non-programmer designers. Second, Takeaway Prototyping focuses on the materialization of lessons learned instead of evolving toward the final system. Iterative prototyping techniques target later stages of the design and are rarely used to explore contexts of use, i.e. they focus on answering questions rather than exploring designs (Buxton, 2007). Within the design space of prototyping tools and techniques, I locate Takeaway Prototyping in-between rapid and iterative prototyping, i.e. faster than iterative techniques and more iterative than rapid techniques (Figure 9.1).

In summary, the main idea behind Takeaway Prototyping is to reify design decisions to support iteration. Even with throwaway prototypes, the abstract concepts that designers learn during rapid prototyping are not thrown away. Designers prototype to explore and to evaluate, but where are the results of that exploration and evaluation? They are carried out to the next iteration only in the memory of the designer or in an external artifact. Instead of forcing designers to keep separate representations of the lessons learned, e.g., in a notebook or with side-comments over the design artifacts, Takeaway Prototyping seeks to reify these ideas as a medium for design within computer-based prototyping tools.

9.3.3 Supporting Takeaway Prototyping

To be able to reuse artifacts first they need to exist as entities in the prototyping tools. Reification is the mechanism of transforming these

	Evolution	Representation	Precision (Low - Medium - High)			Interactivity
			Content	Form	Behavior	
PEN AND PAPER						none
PHYSICAL MOCK-UPS		Paper Prototyping	L	L ⁺	L ⁺	none
WIZARD-OF-OZ						fixed-path / open
VIDEO PROTOTYPING		Rapid Prototyping				none / only watchable
MONTAGE		Prototyping (Throwaway)		-	+	none / only watchable
NON-INTERACTIVE SIMULATION		Software Prototyping	L	MH	M	fixed-path / open
INTERACTIVE SIMULATIONS						fixed-path / open
SCRIPTING LANGUAGES			+		+	
ENACT		Takeaway Prototyping				
GRAPHICAL LIBRARIES						-
WINDOW SYSTEMS		Software Tools	M	MH	LH	open
UI TOOLKITS						
UI BUILDERS		Iterative Prototyping				
APPLICATION FRAMEWORKS		Software Environments	MH	MH	LH	open
MODEL-BASED TOOLS						
UIDES						

Figure 9.1: Extended version of the table presented in Figure 2.13. ENACT embodies most of the principles of Takeaway Prototyping in order to provide rapid as well as iterative prototypes. On the other hand, VIDEOCLIPPER (Video Prototyping) and MONTAGE would need more mechanisms to iterate the prototype to better support Takeaway Prototyping.

objects into manipulable objects. Next, I outline three examples of how reification can help prototyping tools reuse design artifacts to support Takeaway Prototyping.

REIFYING USER NEEDS AS A STORY A story describes a macro task encompassing the user interactions as well as the broader user goal. Describing an interactive system just as a collection of features is a common mistake, as Cooper (1995) says:

"Reducing a product's definition to a list of features and functions ignores the real opportunity — orchestrating technological capability to serve human needs and goals" (Cooper, 1995).

A story concretizes needs and goals for later analysis. However, these story artifacts are rarely part of the prototyping tools. This dissertation presented some examples of how a narrative can be embedded into the prototype itself. For example, video prototypes connect the interface with the story to the point that both are indivisible in the final artifact. Film-making brings a narrative medium to interaction design tools to explain the user's journey. I believe that other tools should embrace this idea of having representations of the interaction but also of the interaction environment.

REIFYING USER INPUTS AS ENACTIONS Design by enaction let designers express interaction with their own body. This communication by demonstration is commonly used within the design teams and

with other stakeholders. However, few tools capture and reify these enactions in the prototyping tool as a design material.

Takeaway Prototyping relies on design by enaction. However, while design by enaction focuses on the micro designer-tool interaction, Takeaway Prototyping outlines a larger process of prototyping. Design by enaction focuses on how the designer should express interactivity, i.e. by acting-out the user's inputs with the actual physical user-computer mediator, while Takeaway Prototyping focuses on using the enactions as the starting point of the prototyping process. This allows designers to start from the concrete and move towards the abstract.

REIFYING EDGE-CASES AS TESTS Reifying the lessons learned helps designers iterate within the prototyping tool. When a design breaks because it did not consider an edge-case, that scenario should be saved in the tool itself for later use. In *ENACT*, the assisted testing lets designers and developers record a particularly difficult user gesture to assess future interaction behaviors over this example. While extending the interaction code, the same scenario can be played automatically to detect regressions. In *MONTAGE*, once a particularly difficult context is recorded, designers can iterate over multiple interfaces independently and later evaluate them in that troublesome situation.

In summary, I designed these prototyping tools to support the early-stage design of dynamic interactions. Many of the features were inspired by the principles of reification, polymorphism, and reuse. To apply these principles, I created several information substrates that are non-visual, visual and interactive. Finally, these tools support a prototyping style that I call Takeaway Prototyping. Takeaway Prototyping materializes the lessons learned to facilitate iteration, embeds the design in a story to outline user needs and supports design by enaction to let designers experience the interaction they are creating.

CONCLUSIONS AND FUTURE WORK

This dissertation addresses the prototyping of interaction in early-stage design. I introduced prototyping as an essential activity in any software process model, even in the phase-oriented waterfall model. According to Houde and Hill (1997), a designer is "anyone who creates a prototype in order to design". Rapid prototyping takes place in early-stage design. It usually produces throwaway prototypes, that can be quickly discarded when they have served their purpose. However, current rapid prototyping tools and techniques poorly support the design of highly interactive systems. First, while paper-based prototypes are inexpensive and easy to produce they lack means to represent dynamic behaviors. Second, the disposability of throwaway prototypes might help reduce design fixation (Jansson and Smith, 1991) but does not support an iterative and incremental process over the design artifacts. In order to address these issues, I presented guidelines and explored computer-based tools to bridge the gap between throwaway prototypes –which are disposable but fast to create– with iterative prototypes –which are slower to create but reusable.

Video is a familiar medium, suitable for early-stage design and flexible enough to accommodate dynamic behaviors. However, I found that video is not as widely used as other prototyping media such as hand-drawn sketches and digital mockups. Designers' main barriers to using video in interaction design are the lack of time or resources to prepare for and record video, as well as to edit video. Video prototyping is a rapid prototyping technique that relies on video as a quick medium to persist, explore and communicate interactive ideas. Video prototyping encourages designers to avoid post-production, thus reducing the need for video editing. By contrast, video prototyping captures video on-the-go, following a previously agreed storyboard.

I developed VIDEOCLIPPER to support this structured capturing of video on-the-go with a flexible and interactive storyboard. Besides planning, VIDEOCLIPPER also supports typical video prototyping activities such as aligning shots or re-designing. VIDEOCLIPPER provides a reusable "ghost image" of the last recorded frame to help designers create invisible cuts. Designers can also reuse complete Storyboards or just Lines during a design iteration to build on top of previously recorded videos. I studied the use of VIDEOCLIPPER during real design activities in several design courses and workshops. All the participants appreciated the simplicity of the tool and most of them were satisfied with the final video prototype. The results suggest that participants spend less time capturing and editing with VIDEOCLIPPER than with

other video tools, such as iMovie. However, to minimize the impact of changing the design, participants rely on ad-hoc methods to avoid re-shooting. For example, they create static reusable backgrounds or they sacrifice the consistency across scenes to avoid reworking.

I developed MONTAGE to reduce re-shooting and increase reuse during video prototyping. MONTAGE decouples the video prototypes into two video streams: the context of use and the user interface. During composition, I use green areas as "the glue" between interface and context videos. This lets designers quickly explore an interface in multiple contexts or multiple interfaces in the same context. MONTAGE re-purposes common smartphones as remote cameras providing an inexpensive setup. The WizardCam captures a paper prototype of the interface, and the UseCam captures the context of use. Both cameras record and stream their content to a central tablet device called the Canvas. On the Canvas, MONTAGE combines these two video streams with digital sketches to create the final video prototype. Designers can directly manipulate these sketches alongside the video playback to create quick and persistent animations. I illustrated the expressive power of MONTAGE with examples of diverse interaction styles: [WIMP](#), touch-based, proxemic, voice and [AR](#)-based interactions.

Video prototyping is a quick and easy technique to let designers illustrate an interactive system in context. However, professional designers and developers often struggle when transitioning between the illustration of the design to the actual implementation of the system. We conducted three studies that focused on the design and implementation of custom interactions to understand the mismatches between designers and developers processes, tools and representations. We found that current practices induce unnecessary rework and cause discrepancies between design and implementation. We identified three recurring types of breakdowns: omitting critical details, ignoring edge cases, and disregarding technical limitations.

We proposed four design principles to create tools that mitigate these problems: *Provide multiple viewpoints, maintain a single source of truth, reveal the invisible* and *support design by enactment*. We applied these principles to create ENACT, an interactive live environment for prototyping touch-based interactions. We conducted two studies to assess ENACT and to compare designer-developer collaboration with ENACT versus current tools. Results suggest that ENACT helps participants detect more edge cases, increases designers' participation and provides new opportunities for co-creation.

10.1 IMPLICATIONS FOR FUTURE RESEARCH

My goal has been to help design teams with concrete prototyping tools that encourage reuse and iteration over previous interaction design artifacts. The three tools presented in this dissertation target each a

different type of prototyping. However, I believe in the potential of combining them. For example, I have started to combine VIDEOCLIPPER and MONTAGE to support planning and composition within one tool.

I explored video and interactive prototyping in isolation but the principles presented in this dissertation can be applied to link both types of prototyping. Currently, ENACT's UI storyboard represents different stages of the systems visual outputs. Designers use digital drawings, demonstrate the user inputs, and associate each storyboard screen with the corresponding input event. Another possibility is to let designers start prototyping in VIDEOCLIPPER. From the created video prototype, one could extract the events from the user-actor's recorded performance and feed them as the inputs to the ENACT's timeline. Also, instead of drawing the storyboard in ENACT with typical WIMP interactions, one could extract meaningful frames from a video prototype created with MONTAGE's WizardCam, e.g., by selecting different frames of the video as the ENACT's storyboard screens. Finally, one could integrate ENACT and MONTAGE the other way around. Designers can start by creating an interactive prototype in ENACT, and then in MONTAGE, it can be embedded into multiple contextual shoots, where the green area is replaced with an interactive prototype.

One of the goals of the tools I developed was the prototyping of custom interactions that do not have established guidelines. However, by materializing common problems and solutions across multiple design teams, interaction design patterns can emerge (Borchers, 2001). These patterns could be integrated in the design tool instead of being just an isolated set of guidelines. For example, in MONTAGE, a set of context-of-use videos could be provided to illustrate users in multiple and common situations interacting with "green areas". In the same way, ENACT could have a collection of common system outputs, recordings of user inputs and even code templates.

I hope that these tools and principles will inspire others to create new prototyping tools to support Takeaway Prototyping. Initiatives such as the Hour of Code (Code.org, 2018) try to spread programming to a broader audience. Undoubtedly, more and more designers will know how to code in the future. However, developer-based representations, such as code or visual programming, are not the only –neither the most adequate– representation for every aspect of an interaction. We need to provide integrated and multiple representations, e.g., symbolic, visual and enactive, to create collaborative spaces in which professionals with different skills, mindsets, and values can work together. Digital tools need to provide ways of navigating these representations in a seamless way, thus reducing reworking and mismatches between design and implementation. I invite other researchers to test and extend these tools and design principles to create new ways to better support the iterative prototyping of interactive systems.

A

APPENDIX

A.1 SURVEY ABOUT VIDEO IN INTERACTION DESIGN

Video to support interaction design

Thank you for answering this questionnaire!

If you are a professional, researcher or student in Interaction Design, your answers will help us develop a new video-based design tool.

* Required

Background

1. Age

Mark only one oval.

- <17
- 18-25
- 26-35
- 36-45
- 46-55
- 56-65
- >65

2. Gender

Mark only one oval.

- Female
- Male
- Other: _____

3. Highest degree

4. Design experience

Mark only one oval.

- <1 year
- 1-3 years
- 4-8 years
- 9-12 years
- >12 years

5. **Current job**

Check all that apply.

- Student
- Professional designer
- Part-time/Freelance designer
- Design researcher
- Other: _____

6. **Job environment**

Check all that apply.

- Academic research
- Corporate research
- Design firm
- Freelance
- Large company
- Unemployed
- Start-up

7. **Country that you work in:**

8. **Please answer the following questions based on either your:**

Mark only one oval.

- Current project
- Most recent completed project

9. **How many design artefacts did you create to explore and express your design?**

Mark only one oval per row.

	None	1-3	4-10	>10
Hand-drawn sketches	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hand-made physical mock-ups	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Computer-drawn sketches	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Computer animations	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Video illustrations	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other tools	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

10. Any other artefacts?

11. How much time did you spend to create these artefacts?

Mark only one oval per row.

	0%	0-20%	20-40%	40-60%	60-80%	80-100%
Hand-drawn sketches	<input type="radio"/>					
Handmade physical mock-ups	<input type="radio"/>					
Computer-drawn sketches	<input type="radio"/>					
Computer animations	<input type="radio"/>					
Video illustrations	<input type="radio"/>					
Other tools	<input type="radio"/>					

12. Please tell us how you used the above artefacts to explore your design:

13. Was this typical? *

Mark only one oval.

- Yes *Skip to question 14.*
- No *Skip to question 15.*

If typical

14. then please describe an unusual example:

for example, you only did coding or drawing for a specific project while you generally go through many ideation process.

Skip to question 16.

If not typical

15. then please describe a typical example:

Skip to question 16.

16. **In general, I use video to:**

Check all that apply.

- to illustrate the step-by-step details of the interaction
- to compare alternative types of interaction
- to figure out how a user moves from one state to another
- I do not use video
- Other: _____

17. **I use video to communicate specific user interactions to ...**

Check all that apply.

- other designers
- software developers
- clients or potential funders
- users
- management
- I do not use video to communicate to others
- Other: _____

18. **I use video to document:**

Check all that apply.

- I do not use video to document my design
- intermediate design stages
- alternative design possibilities
- final design
- Other: _____

19. **Comments or explanations?**

20. **What are the barriers to using video?**

Check all that apply.

- Video is not useful for exploring or expressing the design
- Lack of access to video equipment
- Lack of time or resources to prepare for and record video
- Lack of time or resources to edit video
- Too hard to find relevant video clips
- Video quality is not sufficient
- Other: _____

21. **Which computer tools do you use?**

Check all that apply.

	for this project	in general
I do not use computer tools	<input type="checkbox"/>	<input type="checkbox"/>
Adobe XD	<input type="checkbox"/>	<input type="checkbox"/>
Axure	<input type="checkbox"/>	<input type="checkbox"/>
BALSAMIQ	<input type="checkbox"/>	<input type="checkbox"/>
Framer	<input type="checkbox"/>	<input type="checkbox"/>
Flinto	<input type="checkbox"/>	<input type="checkbox"/>
HTML/CSS	<input type="checkbox"/>	<input type="checkbox"/>
InVision	<input type="checkbox"/>	<input type="checkbox"/>
Illustrator	<input type="checkbox"/>	<input type="checkbox"/>
Indesign	<input type="checkbox"/>	<input type="checkbox"/>
Pixate	<input type="checkbox"/>	<input type="checkbox"/>
Photoshop	<input type="checkbox"/>	<input type="checkbox"/>

22. **List any other computer tools you use.**

23. **How much do you normally pay for interaction design tools?**

Check all that apply.

	I only use free tools	I only use cracked software	Less than 50€	From 50€ to 100€	From 100€ to 200 €	From 200€ to 500€	More than 500€
Annual payment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
One-time payment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

24. **Who usually pays for your design tools?**

Check all that apply.

- Company
- University
- Yourself
- Other institution
- Other: _____

25. **We are interested in any suggestions you have to improve your design tools.**

26. **Other comments?**

27. **Please provide your email if you would like to hear more about our video-based design tool:**

A.2 QUESTIONNAIRES VIDEOCLIPPER

Video Prototyping Questionnaire

This questionnaire will not be part of the course grading

Anonymous Group ID:

White

Yellow

Red

Purple

Maroon

Green

Blue

Black

1. *How many people are in your group?*

2. *Which tool (device + capture software) did you use to **capture** the videos?*

A. Tablet (e.g. iPad + default Camera app): _____

B. Smartphone: _____

C. Other: _____

3. Why did you choose this combination of device + capture software?

4. *How much time did you spend **capturing** the videos?*

How many minutes/hours: _____

_____ % of the whole process (storyboarding+capturing+editing) was dedicated to capturing

_____ people who worked on capturing video

5. *Which tool did you use to **edit** your video prototype?*

A. iMovie

B. Final Cut Pro

C. Other: _____

6. Why did you choose this tool?

7. How much time did you spend **editing** the video?

How many minutes/hours: _____

_____ % of the whole process (storyboarding+capturing+editing) was dedicated to editing

_____ people who worked on editing video

8. Does the % in question 4 and % in question 7 add to 100%? If not, explain

9. Did you shoot the video following the storyboard order?
(only answer this question if you created a storyboard)

A. Yes - We followed the storyboard completely

B. Mostly - We only made a few changes

C. Some - We changed our minds a lot

D. No - We didn't follow the storyboard

Comments:

10. How satisfied are you with the final result? Explain

11. Would VideoClipper have helped you? If so, **how**?
(We are interested in opinions from different people in your group)

12. Any other comment, problems or ideas:

BIBLIOGRAPHY

- Appert, Caroline and Michel Beaudouin-Lafon (2006). "SwingStates: adding state machines to the swing toolkit." In: *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06*. New York, New York, USA: ACM Press, p. 319.
- Apple Inc. (2018a). *AV Foundation - Apple Developer*. <https://developer.apple.com/av-foundation/>. (Visited on 04/04/2018).
- (2018b). *Core Image | Apple Developer Documentation*. <https://developer.apple.com/documentation/coreimage>. (Visited on 04/04/2018).
- Baecker, Ronald M (1969). "Picture-driven Animation." In: *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference*. AFIPS '69 (Spring). New York, NY, USA: ACM, pp. 273–288. DOI: [10.1145/1476793.1476838](https://doi.org/10.1145/1476793.1476838). URL: <http://doi.acm.org/10.1145/1476793.1476838>.
- Bailey, Brian P. and Joseph A. Konstan (2003). "Are informal tools better?: comparing DEMAIS, pencil and paper, and authorware for early multimedia design." In: *Proceedings of the conference on Human factors in computing systems - CHI '03*. New York, New York, USA: ACM Press, p. 313. ISBN: 1581136307. DOI: [10.1145/642611.642666](https://doi.org/10.1145/642611.642666).
- Bailey, Brian P., Joseph A. Konstan, and John V. Carlis (2001). "DEMAIS: Designing Multimedia Applications with Interactive Storyboards." In: *Proceedings of the ninth ACM international conference on Multimedia - MULTIMEDIA '01*. New York, New York, USA: ACM Press, p. 241. ISBN: 1581133944. DOI: [10.1145/500141.500179](https://doi.org/10.1145/500141.500179).
- Balzer, Robert, Cheatham TE Jr, and Cordell Green (1983). "Software Technology in the 1990." In: *Computer* 11, pp. 39–45.
- Bardram, Jakob, Claus Bossen, Andreas Lykke-Olesen, Rune Nielsen, and Kim Halskov Madsen (2002). "Virtual video prototyping of pervasive healthcare systems." In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '02*, p. 167. ISBN: 1581135157. DOI: [10.1145/778712.778738](https://doi.org/10.1145/778712.778738).
- Bardzell, Jeffrey, Shaowen Bardzell, Christian Briggs, Kevin Makice, William Ryan, and Matt Weldon (2006). "Machinima Prototyping: An Approach to Evaluation." In: *Proceedings of the 4th Nordic conference on Human-computer interaction changing roles - NordiCHI '06*. New York, New York, USA: ACM Press, pp. 433–436. ISBN: 1595933255. DOI: [10.1145/1182475.1182530](https://doi.org/10.1145/1182475.1182530).
- Barnes, Connelly, David E. Jacobs, Jason Sanders, Dan B. Goldman, Szymon Rusinkiewicz, Adam Finkelstein, and Maneesh Agrawala (2008). "Video Puppetry: A Performative Interface for Cutout Animation." In: *ACM SIGGRAPH Asia 2008 papers on - SIGGRAPH Asia*

- '08. Vol. 27. 5. New York, New York, USA: ACM Press, p. 1. ISBN: 9781450318310. DOI: [10.1145/1457515.1409077](https://doi.org/10.1145/1457515.1409077).
- Bau, Olivier and Wendy E. Mackay (2008). "OctoPocus: A Dynamic Guide for Learning Gesture-based Command Sets." In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. UIST '08. New York, NY, USA: ACM, pp. 37–46. ISBN: 978-1-59593-975-3. DOI: [10.1145/1449715.1449724](https://doi.org/10.1145/1449715.1449724).
- Bäumer, Dirk, Walter R. Bischofberger, Horst Lichter, and Heinz Zühlighoven (1996). "User Interface Prototyping - Concepts, Tools, and Experience." In: *Proceedings of the 18th International Conference on Software Engineering*. ICSE '96. Washington, DC, USA: IEEE Computer Society, pp. 532–541. ISBN: 0-8186-7246-3.
- Beaudouin-Lafon, Michel (2000). "Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '00. New York, NY, USA: ACM, pp. 446–453. ISBN: 1-58113-216-6. DOI: [10.1145/332040.332473](https://doi.org/10.1145/332040.332473). URL: <http://doi.acm.org/10.1145/332040.332473>.
- (2017). "Towards Unified Principles of Interaction." In: *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*. CHIItaly '17. New York, NY, USA: ACM, 1:1–1:2. ISBN: 978-1-4503-5237-6. DOI: [10.1145/3125571.3125602](https://doi.org/10.1145/3125571.3125602). URL: <http://doi.acm.org/10.1145/3125571.3125602>.
- Beaudouin-Lafon, Michel and Wendy E. Mackay (2000). "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces." In: *Proceedings of the working conference on advanced visual interfaces*, pp. 102–109. DOI: [10.1145/345513.345267](https://doi.org/10.1145/345513.345267).
- (2003). "Prototyping tools and techniques." In: *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pp. 1017–1039. ISBN: 0-8058-3838-4. DOI: [10.1201/9781410615862](https://doi.org/10.1201/9781410615862).
- Beck, Kent (1999). "Embracing change with extreme programming." In: *Computer* 32.10, pp. 70–77.
- Beck, Kent and Alan Cooper (2002). *Extreme Programming vs. Interaction Design*. [http://web.archive.org/web/20020117212245/http://www.fawcette.com/interviews/beck\[_\]cooper/page7.asp](http://web.archive.org/web/20020117212245/http://www.fawcette.com/interviews/beck[_]cooper/page7.asp). (Visited on 08/17/2018).
- Beck, Kent et al. (2001). *Manifesto for Agile Software Development*. <http://www.agilemanifesto.org/>. (Visited on 05/26/2016).
- Benington, H. D. (1983). "Production of Large Computer Programs." In: *Annals of the History of Computing* 5.4, pp. 350–361. ISSN: 0164-1239. DOI: [10.1109/MAHC.1983.10102](https://doi.org/10.1109/MAHC.1983.10102).
- Berning, Matthias, Takuro Yonezawa, Till Riedel, Jin Nakazawa, Michael Beigl, and Hide Tokuda (2013). "pARnorama: 360 Degree Interactive Video for Augmented Reality Prototyping." In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct*

- publication - *UbiComp '13 Adjunct*. New York, New York, USA: ACM Press, pp. 1471–1474. ISBN: 9781450322157. DOI: [10.1145/2494091.2499570](https://doi.org/10.1145/2494091.2499570).
- Berthouzoz, Floraine, Wilmot Li, and Maneesh Agrawala (2012). “Tools for Placing Cuts and Transitions in Interview Video.” In: *ACM Transactions on Graphics* 31.4, pp. 1–8. ISSN: 07300301. DOI: [10.1145/2185520.2185563](https://doi.org/10.1145/2185520.2185563).
- Binder, Thomas (1999). “Setting the stage for improvised video scenarios.” In: *CHI '99 extended abstracts on Human factors in computing systems - CHI '99*. New York, New York, USA: ACM Press, p. 230. ISBN: 1581131585. DOI: [10.1145/632716.632859](https://doi.org/10.1145/632716.632859).
- Bødker, Susanne and Kaj Grønbaek (1991). “Cooperative prototyping: users and designers in mutual activity.” In: *International Journal of Man-Machine Studies* 34.3, pp. 453–478. ISSN: 00207373. DOI: [10.1016/0020-7373\(91\)90030-B](https://doi.org/10.1016/0020-7373(91)90030-B).
- Bødker, Susanne, Pelle Ehn, Dan Sjögren, and Yngve Sundblad (2000). “Co-operative Design—perspectives on 20 years with ‘the Scandinavian IT Design Model’.” In: *proceedings of NordiCHI*. Vol. 2000, pp. 22–24.
- Boehm, Barry (1986). “A spiral model of software development and enhancement.” In: *ACM SIGSOFT Software engineering notes* 11.4, pp. 14–24.
- Bohemian BV (2009). *Sketch*. <https://www.sketchapp.com>.
- Bolchini, Davide, Diego Pulido, and Anthony Faiola (2009). ““Paper in Screen” Prototyping: An Agile Technique to Anticipate the Mobile Experience.” In: *interactions* 16.4, p. 29. ISSN: 10725520. DOI: [10.1145/1551986.1551992](https://doi.org/10.1145/1551986.1551992).
- Bolt, Richard A (1980). “Put-that-there”: *Voice and gesture at the graphics interface*. Vol. 14. 3. ACM.
- Bonanni, Leonardo and Hiroshi Ishii (2009). “Stop-motion prototyping for tangible interfaces.” In: *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09*. New York, New York, USA: ACM Press, p. 315. ISBN: 9781605584935. DOI: [10.1145/1517664.1517729](https://doi.org/10.1145/1517664.1517729).
- Borchers, Jan (2001). *A Pattern Approach to Interaction Design*. Wiley series in software design patterns. John Wiley & Sons, Inc., p. 246. ISBN: 0471498289.
- Bowker, Geoffrey C. and Susan Leigh Star (2000). *Sorting things out: Classification and its consequences*. MIT press.
- Braun, Virginia and Victoria Clarke (2006). “Using thematic analysis in psychology.” In: *Qualitative Research in Psychology* 3.2, pp. 77–101. ISSN: 1478-0887. DOI: [10.1191/1478088706qp0630a](https://doi.org/10.1191/1478088706qp0630a).
- Brooks, Frederick P (1975). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Inc. ISBN: 0-201-00650-2.

- Brooks, Frederick P. and Others (1987). "Defense Science Board Task Force Report on Military Software." In: *Office of the Under Secretary of Defense for Acquisition, Washington, DC* 20301.
- Brown, Judith M., Gitte Lindgaard, and Robert Biddle (2008). "Stories, Sketches, and Lists: Developers and Interaction Designers Interacting Through Artefacts." In: *Agile 2008 Conference*. IEEE, pp. 39–50.
- (2011). "Collaborative Events and Shared Artefacts: Agile Interaction Designers and Developers Working Toward Common Aims." In: *2011 AGILE Conference*. IEEE, pp. 87–96. ISBN: 978-1-61284-426-8. DOI: [10.1109/AGILE.2011.45](https://doi.org/10.1109/AGILE.2011.45).
- (2012). "Joint implicit alignment work of interaction designers and software developers." In: *Proceedings of the 7th Nordic Conference on Human-Computer Interaction Making Sense Through Design - NordiCHI '12*. New York, New York, USA: ACM Press, p. 693.
- Bruner, Jerome S. (2006). "In search of pedagogy: Volume I." In: *New York: Routledge*.
- Buchenau, Marion and Jane Fulton Suri (2000). "Experience prototyping." In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. New York, New York, USA: ACM Press, pp. 424–433. ISBN: 1581132190. DOI: [10.1145/347642.347802](https://doi.org/10.1145/347642.347802).
- Bürdek, Bernhard E. (2005). *Design: History, theory and practice of product design*. Walter de Gruyter.
- Buur, Jacob and Astrid Soendergaard (2000). "Video Card Game: An augmented environment for User Centred Design discussions." In: *Proceedings of DARE 2000 on Designing augmented reality environments - DARE '00*. New York, New York, USA: ACM Press, pp. 63–69. DOI: [10.1145/354666.354673](https://doi.org/10.1145/354666.354673).
- Buxton, Bill (2007). *Sketching User Experiences: getting the design right and the right design*. Morgan Kaufmann, p. 448. ISBN: 978-0123740373. DOI: [10.1016/B978-0-12-374037-3.X5043-3](https://doi.org/10.1016/B978-0-12-374037-3.X5043-3).
- Buxton, John N and Brian Randell (1970). *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee*. NATO Science Committee; available from Scientific Affairs Division, NATO.
- Campos, Pedro and Nuno Nunes (2007). "Practitioner Tools and Workstyles for User-Interface Design." In: *IEEE Software* 24.1, pp. 73–80. ISSN: 0740-7459. DOI: [10.1109/MS.2007.24](https://doi.org/10.1109/MS.2007.24).
- Cardelli, Luca and Peter Wegner (1985). "On understanding types, data abstraction, and polymorphism." In: *ACM Computing Surveys (CSUR)* 17.4, pp. 471–523.
- Carroll, John M (1995). "Scenario-based design: envisioning work and technology in system development." In:
- Carter, Adam S. and Christopher D. Hundhausen (2010). "How is User Interface Prototyping Really Done in Practice? A Survey of User

- Interface Designers." In: *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, pp. 207–211. ISBN: 978-1-4244-7621-3. DOI: [10.1109/VLHCC.2010.36](https://doi.org/10.1109/VLHCC.2010.36).
- Chatty, Stéphane, Stéphane Sire, Jean-Luc Vinot, Patrick Lecoanet, Alexandre Lemort, and Christophe Mertz (2004). "Revisiting visual interface programming." In: *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*. New York, New York, USA: ACM Press, p. 267. ISBN: 1581139578. DOI: [10.1145/1029632.1029678](https://doi.org/10.1145/1029632.1029678).
- Chen, Kevin and Haoqi Zhang (2015). "Remote Paper Prototype Testing." In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. New York, New York, USA: ACM Press, pp. 77–80. ISBN: 9781450331456. DOI: [10.1145/2702123.2702423](https://doi.org/10.1145/2702123.2702423).
- Code.org (2018). *Hour of Code*. <https://hourofcode.com/>. (Visited on 08/01/2018).
- Computer History Museum. *In Your Defense*. <http://www.computerhistory.org/collections/catalog/102651595>. (Visited on 08/17/2018).
- Cooper, Alan (1995). *About Face: The Essentials of User Interface Design*. 1st. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 1568843224.
- (2004). *The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity*. Sams Indianapolis.
- Cooper, Alan, Robert Reimann, and David Cronin (2007). *About face 3: the essentials of interaction design*. Vol. 3. 3. John Wiley & Sons, p. 610. ISBN: 9780470084113. DOI: [10.1057/palgrave.ivs.9500066](https://doi.org/10.1057/palgrave.ivs.9500066).
- Dahl, O (1970). "Simula 67 common base language." In: *Technical report*, S–22.
- Dahl, Ole-Johan, Edsger Wybe Dijkstra, and Charles Antony Richard Hoare (1972). *Structured programming*. Academic Press Ltd.
- Davis, Richard C., Brien Colwell, and James A. Landay (2008). "K-Sketch: A "Kinetic" Sketch Pad for Novice Animators." In: *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. New York, New York, USA: ACM Press, p. 413. ISBN: 9781605580111. DOI: [10.1145/1357054.1357122](https://doi.org/10.1145/1357054.1357122).
- Dhillon, Beant, Peter Banach, Rafal Kocielnik, Jorge Peregrín Emparanza, Ioannis Politis, Agata Raczewska, and Panos Markopoulos (2011). "Visual Fidelity of Video Prototypes and User Feedback: A Case Study." In: *Proceedings of the 25th BCS Conference on Human-Computer Interaction*. BCS-HCI '11. Swinton, UK, UK: British Computer Society, pp. 139–144.
- Djajadiningrat, J. P., William Gaver, and J. W. Fres (2000). "Interaction Relabelling and Extreme Characters: Methods for Exploring Aesthetic Interactions." In: *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*. DIS '00. New York, NY, USA: ACM, pp. 66–71. ISBN: 1-58113-219-0. DOI: [10.1145/347642.347664](https://doi.org/10.1145/347642.347664).

- Dubberly, Hugh and D Mitsch (1992). "Knowledge navigator." In: ACM SIGCHI. URL: <http://www.dubberly.com/articles/the-making-of-knowledge-navigator.html>.
- Erickson, Thomas (2000). "Lingua Francas for design." In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. New York, New York, USA: ACM Press, pp. 357–368. ISBN: 1581132190. DOI: [10.1145/347642.347794](https://doi.org/10.1145/347642.347794).
- Evans, Harold, Gail Buckland, and David Lefer (2009). *They made America: from the steam engine to the search engine: two centuries of innovators*. Hachette UK.
- Facebook (2013). *Origami*. <https://origami.design/>.
- Fernaes, Ylva and Petra Sundström (2012). "The material move how materials matter in interaction design research." In: *Proceedings of the Designing Interactive Systems Conference on - DIS '12*. New York, New York, USA: ACM Press, p. 486. ISBN: 9781450312103. DOI: [10.1145/2317956.2318029](https://doi.org/10.1145/2317956.2318029).
- Ferreira, Jennifer, Helen Sharp, and Hugh Robinson (2011). "User Experience Design and Agile Development: Managing Cooperation Through Articulation Work." In: *Softw. Pract. Exper.* 41.9, pp. 963–974. ISSN: 0038-0644. DOI: [10.1002/spe.1012](https://doi.org/10.1002/spe.1012).
- Flinto (2010). *Flinto*. <https://www.flinto.com/>.
- Floyd, Christiane (1984). "A systematic look at prototyping." In: *Approaches to prototyping*. Springer, pp. 1–18.
- Forsyth, Jason B. and Tom L. Martin (2014). "Extracting behavioral information from electronic storyboards." In: *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems - EICS '14*. New York, New York, USA: ACM Press, pp. 253–262. ISBN: 9781450327251. DOI: [10.1145/2607023.2607034](https://doi.org/10.1145/2607023.2607034).
- Fouse, Adam, Nadir Weibel, Edwin Hutchins, and James D. Hollan (2011). "ChronoViz: A System for Supporting Navigation of Time-coded Data." In: *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*. New York, New York, USA: ACM Press, p. 299. ISBN: 9781450302685. DOI: [10.1145/1979742.1979706](https://doi.org/10.1145/1979742.1979706).
- Framer BV (2015). *Framer*. <https://framer.com/>.
- Freeman, Dustin E.R., Stephanie Santosa, Fanny Chevalier, Ravin Balakrishnan, Karan Singh, Dustin E.R. Freeman, Stephanie Santosa, Fanny Chevalier, Ravin Balakrishnan, and Karan Singh (2014). "LACES: Live Authoring through Compositing and Editing of Streaming Video." In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, pp. 1207–1216. ISBN: 9781450324731. DOI: [10.1145/2556288.2557304](https://doi.org/10.1145/2556288.2557304).
- Freeman, Dustin and Ravin Balakrishnan (2016). "Improv Remix: Mixed-Reality Video Manipulation Using Whole-Body Interaction to Extend Improvised Theatre." In: *Proceedings of the 2016 ACM*

- Conference on Designing Interactive Systems - DIS '16*. New York, New York, USA: ACM Press, pp. 533–542. ISBN: 9781450340311. DOI: [10.1145/2901790.2901894](https://doi.org/10.1145/2901790.2901894).
- Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.
- Gandy, Maribeth and Blair MacIntyre (2014). “Designer’s augmented reality toolkit, ten years later.” In: *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, pp. 627–636. ISBN: 9781450330695. DOI: [10.1145/2642918.2647369](https://doi.org/10.1145/2642918.2647369).
- Garcia, Jérémie, Theophanis Tsandilas, Carlos Agon, and Wendy Mackay (2012). “Interactive Paper Substrates to Support Musical Creation.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. New York, NY, USA: ACM, pp. 1825–1828. ISBN: 978-1-4503-1015-4. DOI: [10.1145/2207676.2208316](https://doi.org/10.1145/2207676.2208316). URL: <http://doi.acm.org/10.1145/2207676.2208316>.
- Garcia, Jérémie, Theophanis Tsandilas, Carlos Agon, and Wendy E. Mackay (2014). “Structured observation with polyphony.” In: *Proceedings of the 2014 conference on Designing interactive systems - DIS '14*. New York, New York, USA: ACM Press, pp. 199–208. ISBN: 9781450329026. DOI: [10.1145/2598510.2598512](https://doi.org/10.1145/2598510.2598512).
- Girgensohn, Andreas, John Boreczky, Patrick Chiu, John Doherty, Jonathan Foote, Gene Golovchinsky, Shingo Uchihashi, and Lynn Wilcox (2000). “A semi-automatic approach to home video editing.” In: *Proceedings of the 13th annual ACM symposium on User interface software and technology - UIST '00*. New York, New York, USA: ACM Press, pp. 81–89. ISBN: 1581132123. DOI: [10.1145/354401.354415](https://doi.org/10.1145/354401.354415).
- Glass, Robert L (2006). “The Standish report: does it really describe a software crisis?” In: *Communications of the ACM* 49.8, pp. 15–16.
- Goldberg, Adele and David Robson (1983). *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc.
- Goodman, Danny (1998). *The complete HyperCard 2.2 handbook*. iUniverse.com, Incorporated.
- Gordon, V.S. and J.M. Bieman (1995). “Rapid prototyping: lessons learned.” In: *IEEE Software* 12.1, pp. 85–95. ISSN: 0740-7459. DOI: [10.1109/52.363162](https://doi.org/10.1109/52.363162).
- Green, Paul and Lisa Wei-Haas (1985). “The Rapid Development of User Interfaces: Experience with the Wizard of OZ Method.” In: *Proceedings of the Human Factors Society Annual Meeting* 29.5, pp. 470–474. DOI: [10.1177/154193128502900515](https://doi.org/10.1177/154193128502900515).
- Greenbaum, Joan (2017). “A design of one’s own: towards participatory design in the United States.” In: *Participatory Design*. CRC Press, pp. 27–37.

- Greenberg, Saul, Nicolai Marquardt, Till Ballendat, Rob Diaz-Marino, and Miaosen Wang (2011). "Proxemic Interactions: The New Ubi-comp?" In: *interactions* 18.1, pp. 42–50. ISSN: 1072-5520. DOI: [10.1145/1897239.1897250](https://doi.org/10.1145/1897239.1897250).
- Greenberg, Saul, Carpendale Sheelagh, Marquardt Nicolai, and Buxton Bill (2012). *Sketching User Experiences: The Workbook*. Morgan Kaufmann, p. 272. ISBN: 978-0-12-381959-8. DOI: [10.1016/C2009-0-61147-8](https://doi.org/10.1016/C2009-0-61147-8).
- Grigoreanu, Valentina, Roland Fernandez, Kori Inkpen, and George Robertson (2009). "What designers want: Needs of interactive application designers." In: *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, pp. 139–146.
- Guggenheim, Michael (2010). "The long history of prototypes." In: *Limn* 1.
- Hall, Roger R. (2001). "Prototyping for usability of new technology." In: *International Journal of Human-Computer Studies* 55.4, pp. 485–501. ISSN: 1071-5819. DOI: [10.1006/IJHC.2001.0478](https://doi.org/10.1006/IJHC.2001.0478).
- Halskov, Kim and Rune Nielsen (2006). "Virtual Video Prototyping." In: *Human-Computer Interaction* 21.2, pp. 199–233. ISSN: 0737-0024. DOI: [10.1207/s15327051hci2102_2](https://doi.org/10.1207/s15327051hci2102_2).
- Harmand, Sonia et al. (2015). "3.3-million-year-old stone tools from Lomekwi 3, West Turkana, Kenya." In: *Nature* 521.7552, p. 310.
- Hartmann, Björn (2009). "Gaining Design Insight Through Interaction Prototyping Tools." PhD thesis. Stanford University. ISBN: 978-1-109-44552-7.
- Hartmann, Björn, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee (2006). "Reflective physical prototyping through integrated design, test, and analysis." In: *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06*. New York, New York, USA: ACM Press, p. 299. ISBN: 1595933131. DOI: [10.1145/1166253.1166300](https://doi.org/10.1145/1166253.1166300).
- Hartmann, Björn, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer (2008). "Design as Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning." In: *Proceedings of the 21st annual ACM symposium on User interface software and technology - UIST '08*. New York, New York, USA: ACM Press, p. 91. ISBN: 9781595939753. DOI: [10.1145/1449715.1449732](https://doi.org/10.1145/1449715.1449732).
- Hertzfeld, Andy (1982). *Folklore.org: Calculator Construction Set*. https://www.folklore.org/StoryView.py?story=Calculator_{_}Construction_{_}Set.txt. (Visited on 08/17/2018).
- Holmquist, Lars Erik (Mar. 2005). "Prototyping: Generating Ideas or Cargo Cult Designs?" In: *Interactions* 12.2, pp. 48–54. ISSN: 1072-5520. DOI: [10.1145/1052438.1052465](https://doi.org/10.1145/1052438.1052465). URL: <http://doi.acm.org/10.1145/1052438.1052465>.

- Houde, Stephanie and Charles Hill (1997). *What do Prototypes Prototype?* Tech. rep. Apple Computer, Inc.
- Hunt, Andrew and David Thomas (1999). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional. ISBN: 9780201616224. URL: <https://www.amazon.com/Pragmatic-Programmer-Journeyman-Master/dp/020161622X?SubscriptionId=AKIAI0BINVZYXZQZ2U3A&tag=chimbiori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=020161622X>.
- Jalal, Ghita, Nolwenn Maudet, and Wendy E. Mackay (2015). "Color Portraits: From Color Picking to Interacting with Color." In: *Proceedings of the ACM CHI'15 Conference on Human Factors in Computing Systems*. Vol. 1, pp. 4207–4216.
- Janson, Marius A. and L. Douglas Smith (1985). "Prototyping for Systems Development: A Critical Appraisal." In: *MIS Q.* 9.4, pp. 305–316. ISSN: 0276-7783. DOI: [10.2307/249231](https://doi.org/10.2307/249231).
- Jansson, David G. and Steven M. Smith (1991). "Design fixation." In: *Design Studies* 12.1, pp. 3–11. ISSN: 0142-694X. DOI: [https://doi.org/10.1016/0142-694X\(91\)90003-F](https://doi.org/10.1016/0142-694X(91)90003-F).
- Johnson, Jeff, ed. (2000). *GUI Bloopers: Don'Ts and Do's for Software Developers and Web Designers*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1-55860-582-7.
- Jokela, Tero, Minna Karukka, and Kaj Mäkelä (2007). "Mobile Video Editor: Design and Evaluation." In: *Human-Computer Interaction. Interaction Platforms and Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 344–353. DOI: [10.1007/978-3-540-73107-8_39](https://doi.org/10.1007/978-3-540-73107-8_39).
- Kato, Jun, Takeo Igarashi, and Masataka Goto (2016). "Programming with Examples to Develop Data-Intensive User Interfaces." In: *Computer* 49.7, pp. 34–42. ISSN: 0018-9162. DOI: [10.1109/MC.2016.217](https://doi.org/10.1109/MC.2016.217).
- Kay, Alan (1984). "Computer Software." In: *Scientific American* 251.3, pp. 52–59. ISSN: 0036-8733. DOI: [10.1038/scientificamerican0984-52](https://doi.org/10.1038/scientificamerican0984-52).
- Kazi, Rubaiat Habib, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice (2014). "Kitty: Sketching Dynamic and Interactive Illustrations." In: *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, pp. 395–405. ISBN: 9781450330695. DOI: [10.1145/2642918.2647375](https://doi.org/10.1145/2642918.2647375).
- Khella Productions Inc (2013). *Keynotopia*. <https://keynotopia.com/>.
- Kim, Han-Jong, Ju-Whan Kim, and Tek-Jin Nam (2016). "miniStudio: Designers' Tool for Prototyping Ubicomp Space with Interactive Miniature." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. New York, New York, USA: ACM Press, pp. 213–224. ISBN: 9781450333627. DOI: [10.1145/2858036.2858180](https://doi.org/10.1145/2858036.2858180).

- Kim, Joy, Mira Dontcheva, Wilmot Li, Michael S. Bernstein, and Daniela Steinsapir (2015). "Motif: Supporting Novice Creativity through Expert Patterns." In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. New York, New York, USA: ACM Press, pp. 1211–1220. ISBN: 9781450331456. DOI: [10.1145/2702123.2702507](https://doi.org/10.1145/2702123.2702507).
- Kim, Ju-Whan and Tek-Jin Nam (2013). "EventHurdle: Supporting Designers' Exploratory Interaction Prototyping with Gesture-Based Sensors." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. New York, New York, USA: ACM Press, p. 267. ISBN: 9781450318990. DOI: [10.1145/2470654.2470691](https://doi.org/10.1145/2470654.2470691).
- Kin, Kenrick (2012). "Investigating the Design and Development of Multitouch Applications." PhD thesis.
- Kin, Kenrick, Björn Hartmann, Tony DeRose, and Maneesh Agrawala (2012a). "Proton++: A Customizable Declarative Multitouch Framework." In: *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, p. 477. ISBN: 9781450315807. DOI: [10.1145/2380116.2380176](https://doi.org/10.1145/2380116.2380176).
- (2012b). "Proton: Multitouch Gestures as Regular Expressions." In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. New York, New York, USA: ACM Press, p. 2885. ISBN: 9781450310154. DOI: [10.1145/2207676.2208694](https://doi.org/10.1145/2207676.2208694).
- Klokose, Clemens N, James R Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon (2015). "Webstrates: Shareable Dynamic Media." In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15. New York, NY, USA: ACM, pp. 280–290. ISBN: 978-1-4503-3779-3. DOI: [10.1145/2807442.2807446](https://doi.org/10.1145/2807442.2807446). URL: <http://doi.acm.org/10.1145/2807442.2807446>.
- Krasner, Glenn E. and Stephen T. Pope (1988). "A description of the model-view-controller user interface paradigm in the smalltalk-80 system." In: *Journal of object oriented programming* 1.3, pp. 26–49.
- Landay, James A. and Brad A. Myers (1995). "Interactive Sketching for the Early Stages of User Interface Design." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '95. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., pp. 43–50. ISBN: 0-201-84705-1. DOI: [10.1145/223904.223910](https://doi.org/10.1145/223904.223910).
- Ledo, David, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg (2018). "Evaluation Strategies for HCI Toolkit Research." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ISBN: 9781450356206. DOI: [10.1145/3173574.3173610](https://doi.org/10.1145/3173574.3173610).
- Lee, Sang Won, Yujin Zhang, Isabelle Wong, Yiwei Yang, Stephanie D. O'Keefe, and Walter S. Lasecki (2017). "SketchExpress: Remixing Animations for More Effective Crowd-Powered Prototyping of

- Interactive Interfaces." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17*. New York, New York, USA: ACM Press, pp. 817–828. ISBN: 9781450349819. DOI: [10.1145/3126594.3126595](https://doi.org/10.1145/3126594.3126595).
- Lelie, Corrie van der (2006). "The value of storyboards in the product design process." In: *Personal and Ubiquitous Computing* 10.2-3, pp. 159–162. ISSN: 1617-4909. DOI: [10.1007/s00779-005-0026-7](https://doi.org/10.1007/s00779-005-0026-7). URL: <http://link.springer.com/10.1007/s00779-005-0026-7>.
- Li, Wenbin, Fabio Viola, Jonathan Starck, Gabriel J. Brostow, and Neill D. F. Campbell (2016). "Roto++: Accelerating Professional Rotoscoping using Shape Manifolds." In: *ACM Transactions on Graphics* 35.4, pp. 1–15. ISSN: 07300301. DOI: [10.1145/2897824.2925973](https://doi.org/10.1145/2897824.2925973).
- Li, Yang and James A. Landay (2005). "Informal prototyping of continuous graphical interactions by demonstration." In: *Proceedings of the 18th annual ACM symposium on User interface software and technology - UIST '05*. New York, New York, USA: ACM Press, p. 221. ISBN: 1595932712. DOI: [10.1145/1095034.1095071](https://doi.org/10.1145/1095034.1095071).
- Li, Yang, Xiang Cao, Katherine Everitt, Morgan Dixon, and James A. Landay (2010). "FrameWire : A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests." In:
- Lim, Youn-Kyung, Erik Stolterman, and Josh Tenenber (2008). "The Anatomy of Prototypes: Prototypes as Filters, Prototypes as Manifestations of Design Ideas." In: *ACM Transactions on Computer-Human Interaction* 15.2, pp. 1–27. ISSN: 10730516. DOI: [10.1145/1375761.1375762](https://doi.org/10.1145/1375761.1375762).
- Lim, Youn-kyung, Apurva Pangam, Subashini Periyasami, and Shweta Aneja (2006). "Comparative Analysis of High- and Low-fidelity Prototypes for More Valid Usability Evaluations of Mobile Devices." In: *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles*. NordiCHI '06. New York, NY, USA: ACM, pp. 291–300. ISBN: 1-59593-325-5. DOI: [10.1145/1182475.1182506](https://doi.org/10.1145/1182475.1182506).
- Loewy, Raymond (2002). *Never leave well enough alone*. JHU Press.
- Löwgren, Jonas (1995). "Applying design methodology to software development." In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, & techniques - DIS '95*. New York, New York, USA: ACM Press, pp. 87–95. ISBN: 0897916735. DOI: [10.1145/225434.225444](https://doi.org/10.1145/225434.225444).
- Löwgren, Jonas and Jonas (2004). "Animated Use Sketches as Design Representations Interaction." In: *interactions* 11.6, p. 22. ISSN: 10725520. DOI: [10.1145/1029036.1029048](https://doi.org/10.1145/1029036.1029048).
- Luciani, Danwei Tran and Peter Vistisen (2017). "Empowering Non-Designers Through Animation-based Sketching." In: 7.
- MacIntyre, Blair, Maribeth Gandy, Steven Dow, and Jay David Bolter (2004). "DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences Blair." In: *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*.

- New York, New York, USA: ACM Press, p. 197. ISBN: 1581139578. DOI: [10.1145/1029632.1029669](https://doi.org/10.1145/1029632.1029669).
- Mackay, Wendy E. (1988). *Video Prototyping: a technique for developing hypermedia systems*. Vol. 5. ACM/SIGCHI.
- Mackay, Wendy E (2002). *Using video to support interaction design*. . Directed by C. Leininger. Video Tutorial distributed at CHI '02. INRIA Multimedia Services.
- Mackay, Wendy E. and Michel Beaudouin-Lafon (1998). "DIVA: Exploratory Data Analysis with Multimedia Streams." In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '98*. New York, New York, USA: ACM Press, pp. 416–423. ISBN: 0201309874. DOI: [10.1145/274644.274701](https://doi.org/10.1145/274644.274701). URL: <http://portal.acm.org/citation.cfm?doid=274644.274701>.
- Mackay, Wendy E. and Glorianna Davenport (1989). "Virtual video editing in interactive multimedia applications." In: *Communications of the ACM* 32.7, pp. 802–810. ISSN: 00010782. DOI: [10.1145/65445.65447](https://doi.org/10.1145/65445.65447).
- Mackay, Wendy E. and Anne-Laure Fayard (1997). "HCI, Natural Science and Design: A Framework for Triangulation Across Disciplines." In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '97*. New York, New York, USA: ACM Press, pp. 223–234. ISBN: 0897918630. DOI: [10.1145/263552.263612](https://doi.org/10.1145/263552.263612).
- (1999). "Video brainstorming and prototyping: techniques for participatory design." In: *CHI'99 extended abstracts on Human factors in ...* May, pp. 118–119. DOI: [10.1145/632716.632790](https://doi.org/10.1145/632716.632790).
- Mackay, Wendy E. and Daniele S. Pagani (1994). "Video mosaic: laying out time in a physical space." In: *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pp. 165–172. DOI: <http://doi.acm.org/10.1145/192593.192646>.
- Mackay, Wendy E., Anne V. Ratzer, and Paul Janecek (2000). "Video artifacts for design: bridging the Gap between abstraction and detail." In: *DIS '00*. New York, New York, USA: ACM, pp. 72–82. ISBN: 978-1-58113-219-9. DOI: [10.1145/347642.347666](https://doi.org/10.1145/347642.347666).
- Mackay, Wendy E., R. Guindon, M. M. Mantel, Lucy Suchman, and D. G. Tatar (1988). "Video: Data for Studying Human-Computer Interaction." In: *Chi'88*, pp. 133–137. DOI: <http://doi.acm.org/10.1145/57167.57189>.
- Marvel Prototyping Ltd (2012). *POP*. <https://marvelapp.com/pop/>.
- Maudet, Nolwenn, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E Mackay (2017a). "Beyond Grids: Interactive Graphical Substrates to Structure Digital Layout." In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. New York, NY, USA: ACM, pp. 5053–5064. ISBN: 978-1-4503-4655-9. DOI: [10.1145/3025453.3025718](https://doi.org/10.1145/3025453.3025718). URL: <http://doi.acm.org/10.1145/3025453.3025718>.

- Maudet, Nolwenn, Germán Leiva, Michel Beaudouin-Lafon, and Wendy E. Mackay (2017b). "Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems." In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing - CSCW '17*. The first two authors contributed equally to this work., pp. 630–641. DOI: [10.1145/2998181.2998190](https://doi.org/10.1145/2998181.2998190). URL: <http://dl.acm.org/citation.cfm?doid=2998181.2998190>.
- McCloud, Scott (1993). "Understanding comics: The invisible art." In: *Northampton, Mass.*
- McCracken, Daniel D and Michael A Jackson (1982). "Life cycle concept considered harmful." In: *ACM SIGSOFT Software Engineering Notes 7.2*, pp. 29–32.
- McCurdy, Michael, Christopher Connors, Guy Pyrzak, Bob Kanefsky, and Alonso Vera (2006). "Breaking the Fidelity Barrier: An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success." In: *Proceedings of the SIGCHI conference on Human Factors in computing systems - CHI '06*, p. 1233. DOI: [10.1145/1124772.1124959](https://doi.org/10.1145/1124772.1124959).
- Moffett, Jack (2014). *Bridging UX and Web Development: Better Results through Team Integration*. Elsevier Science, p. 224.
- Moggridge, Bill and Bill Atkinson (2007). *Designing interactions*. Vol. 17. MIT press Cambridge, MA.
- Muller, Michael J. (1991). "PICTIVE—an exploration in participatory design." In: *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91*. New York, New York, USA: ACM Press, pp. 225–231. ISBN: 0897913833. DOI: [10.1145/108844.108896](https://doi.org/10.1145/108844.108896).
- Muller, Michael J. and Sarah Kuhn (1993). "Participatory design." In: *Communications of the ACM 36.6*, pp. 24–28. ISSN: 00010782. DOI: [10.1145/153571.255960](https://doi.org/10.1145/153571.255960).
- Myers, Brad A., Scott E. Hudson, and Randy Pausch (2000). "Past, present, and future of user interface software tools." In: *ACM Transactions on Computer-Human Interaction (TOCHI) 7.1*, pp. 3–28. ISSN: 10730516. DOI: [10.1145/344949.344959](https://doi.org/10.1145/344949.344959).
- Myers, Brad A., Richard G. McDaniel, and David Wolber (2000). "Programming by example: intelligence in demonstrational interfaces." In: *Communications of the ACM 43.3*, pp. 82–89. ISSN: 00010782. DOI: [10.1145/330534.330545](https://doi.org/10.1145/330534.330545).
- Myers, Brad A., Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew J. Ko (2008). "How designers design and program interactive behaviors." In: *Proceedings - 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008*, pp. 177–184. ISSN: 1943-6092. DOI: [10.1109/VLHCC.2008.4639081](https://doi.org/10.1109/VLHCC.2008.4639081).
- Newman, Mark W. and James A. Landay (2000). "Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice." In:

- Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. New York, New York, USA: ACM Press, pp. 263–274.
- Nielsen, Jakob (1994). *Usability engineering*. Elsevier.
- Norman, Don and Stephen W. Draper (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc. ISBN: 0898597811.
- Oney, Stephen, Brad A. Myers, and Joel Brandt (2014). “InterState: A Language and Environment for Expressing Interface Behavior.” In: *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, pp. 263–272. ISBN: 9781450330695. DOI: [10.1145/2642918.2647358](https://doi.org/10.1145/2642918.2647358).
- Online Etymology Dictionary (2018a). *design* | Origin and meaning of design. <https://www.etymonline.com/word/design>. (Visited on 08/17/2018).
- (2018b). *prototype* | Origin and meaning of prototype. <https://www.etymonline.com/word/prototype>. (Visited on 08/17/2018).
- Orlikowski, Wanda J (1992). “The duality of technology: Rethinking the concept of technology in organizations.” In: *Organization science* 3.3, pp. 398–427.
- Osborn, Alex Faickney (1963). *Applied Imagination: Principles and Procedures of Creative Problem-Solving: Third Revised Edition*. Scribner.
- Osterwalder, Alexander, Yves Pigneur, Gregory Bernarda, and Alan Smith (2014). *Value proposition design: How to create products and services customers want*. John Wiley & Sons.
- Ozenc, Fatih Kursat, Miso Kim, John Zimmerman, Stephen Oney, and Brad A. Myers (2010). “How to support designers in getting hold of the immaterial material of software.” In: *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. New York, New York, USA: ACM Press, p. 2513. ISBN: 9781605589299. DOI: [10.1145/1753326.1753707](https://doi.org/10.1145/1753326.1753707).
- Park, Sun Young, Brad A. Myers, and Andrew J. Ko (2008). “Designers’ natural descriptions of interactive behaviors.” In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, pp. 185–188.
- Pavel, Amy, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala (2015). “SceneSkim: Searching and Browsing Movies Using Synchronized Captions, Scripts and Plot Summaries.” In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. New York, New York, USA: ACM Press, pp. 181–190. ISBN: 9781450337793. DOI: [10.1145/2807442.2807502](https://doi.org/10.1145/2807442.2807502).
- Poltrock, Steven E. and Jonathan Grudin (1994). “Organizational Obstacles to Interface and Development: Two Participant – Observer Studies.” In: *ACM Trans. Comput.-Hum. Interact.* 1.1, pp. 52–80. DOI: [10.1145/174630.174633](https://doi.org/10.1145/174630.174633).

- Rettig, Marc (1994). "Prototyping for tiny fingers." In: *Communications of the ACM* 37.4, pp. 21–27. ISSN: 00010782. DOI: [10.1145/175276.175288](https://doi.org/10.1145/175276.175288).
- Rowland, Ingrid D., Thomas Noble Howe, and Others (2001). *Vitruvius: Ten Books on Architecture*. Cambridge University Press.
- Royce, Winston W (1987). "Managing the development of large software systems: concepts and techniques." In: *Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, pp. 328–338.
- Rudd, Jim and Scott Isensee (1994). "Twenty-two tips for a happier, healthier prototype." In: *interactions* 1.1, pp. 35–40. ISSN: 10725520. DOI: [10.1145/174800.174804](https://doi.org/10.1145/174800.174804).
- Rudd, Jim, Ken Stern, and Scott Isensee (1996). "Low vs. High-fidelity Prototyping Debate." In: *interactions* 3.1, pp. 76–85. ISSN: 1072-5520. DOI: [10.1145/223500.223514](https://doi.org/10.1145/223500.223514).
- Sá, Marco de, Judd Antin, David Shamma, and Elizabeth F. Churchill (2011). "Mobile Augmented Reality: Video Prototyping." In: *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*. New York, New York, USA: ACM Press, p. 1897. ISBN: 9781450302685. DOI: [10.1145/1979742.1979927](https://doi.org/10.1145/1979742.1979927).
- Salah, Dina, Richard F. Paige, and Paul Cairns (2014). "A systematic literature review for agile development processes and user centred design integration." In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. New York, New York, USA: ACM Press, pp. 1–10.
- Santosa, Stephanie, Fanny Chevalier, Ravin Balakrishnan, and Karan Singh (2013). "Direct space-time trajectory control for visual media editing." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. New York, New York, USA: ACM Press, p. 1149. ISBN: 9781450318990. DOI: [10.1145/2470654.2466148](https://doi.org/10.1145/2470654.2466148).
- Schachman, Toby (2017). *Apparatus: A Hybrid Graphics Editor / Programming Environment for Creating Interactive Diagrams*.
- Shaw, Susannah (2017). *Stop motion: craft skills for model animation*. Focal press.
- Shipman, Frank M. and Catherine C. Marshall (1999). "Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems." In: *Computer Supported Cooperative Work (CSCW)* 8.4, pp. 333–352. ISSN: 0925-9724.
- Shneiderman, Ben (1983). "Direct Manipulation: A Step Beyond Programming Languages." In: *Computer* 16.8, pp. 57–69. ISSN: 0018-9162. DOI: [10.1109/MC.1983.1654471](https://doi.org/10.1109/MC.1983.1654471).
- Silva, Paulo Pinheiro da and Norman W. Paton (2000). "UMLi: the unified modeling language for interactive applications." PhD thesis. University of Manchester, pp. 117–132.

- Silva, Thiago Rocha, Jean-Luc Hak, Marco Winckler, and Olivier Nicolas (2017). "A Comparative Study of Milestones for Featuring GUI Prototyping Tools." In: *Journal of Software Engineering and Applications* 10.06, pp. 564–589. ISSN: 1945-3116. DOI: [10.4236/jsea.2017.106031](https://doi.org/10.4236/jsea.2017.106031).
- Silva, Tiago da, Angela Martin, Frank Maurer, and Milene Selbach Silveira (2011). "User-Centered Design and Agile Methods: A Systematic Review." English. In: *2011 AGILE Conference*. IEEE, pp. 77–86.
- Smith, David Canfield, Charles Irby, Ralph Kimball, Bill Verplank, and Eric Harslem (1982). "Designing the Star user interface." In: *Byte* 7, pp. 242–282.
- Snyder, Carolyn (2004). *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, p. 408. ISBN: 978-1-55860-870-2. DOI: [10.1016/B978-1-55860-870-2.X5023-2](https://doi.org/10.1016/B978-1-55860-870-2.X5023-2).
- Sokoler, Tomas and Håkan Edeholt (2002). "Physically embodied video snippets supporting collaborative exploration of video material during design sessions." In: *Proceedings of the second Nordic conference on Human-computer interaction - NordiCHI '02*, p. 139. DOI: [10.1145/572020.572037](https://doi.org/10.1145/572020.572037).
- Sparke, Penny (2009). *The genius of design*. Quadrille London.
- Stansky, Peter (1985). *Redesigning the world: William Morris, the 1880s, and the Arts and Crafts*. Vol. 6. Princeton University Press Princeton, NJ.
- Star, Susan Leigh (1989). "The Structure of Ill-structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving." In: *Distributed Artificial Intelligence (Vol. 2)*. Ed. by M. Huhns. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 37–54. ISBN: 0-273-08810-6.
- (2010). "This is Not a Boundary Object: Reflections on the Origin of a Concept." In: *Science, Technology, & Human Values* 35.5, pp. 601–617. ISSN: 0162-2439. DOI: [10.1177/0162243910377624](https://doi.org/10.1177/0162243910377624).
- Star, Susan Leigh and James R. Griesemer (1989). "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39." In: *Social studies of science* 19.3, pp. 387–420. ISSN: 03063127.
- Stolterman, Erik, Jamie McAtee, David Royer, and Selvan Thandapani (2009). "Designerly Tools." In:
- Tanimoto, Steven L (2013). "A Perspective on the Evolution of Live Programming." In: *Proceedings of the 1st International Workshop on Live Programming*. LIVE '13. Piscataway, NJ, USA: IEEE Press, pp. 31–34. ISBN: 978-1-4673-6265-8.
- Thacker, Charles P, E. M. MacCreight, and Butler W. Lampson (1979). *Alto: A personal computer*. Xerox, Palo Alto Research Center Palo Alto.

- The American Society of Mechanical Engineering (2018). *Top 4 Ancient Design Prototypes*. <https://www.asme.org/engineering-topics/articles/manufacturing-design/top-4-ancient-design-prototypes>. (Visited on 08/17/2018).
- Thompson, Michael and Nina Wishbow (1992). "Prototyping: tools and techniques." In: *Proceedings of the 10th annual international conference on Systems documentation - SIGDOC '92*. New York, New York, USA: ACM Press, pp. 191–199. ISBN: 0897915321. DOI: [10.1145/147001.147030](https://doi.org/10.1145/147001.147030).
- Tognazzini, Bruce (1994). "The "Starfire" video prototype project." In: *Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94*. New York, New York, USA: ACM Press, pp. 99–105. ISBN: 0897916506. DOI: [10.1145/191666.191712](https://doi.org/10.1145/191666.191712).
- United States. Navy Mathematical Computing Advisory Panel (1956). *Symposium on Advanced Programming Methods for Digital Computers: Washington, D.C., June 28, 29, 1956*. ONR symposium report. Office of Naval Research, Department of the Navy.
- Vertelney, Laurie (1989). "Using video to prototype user interfaces." In: *ACM SIGCHI Bulletin* 21.2, pp. 57–61. ISSN: 07366906. DOI: [10.1145/70609.70615](https://doi.org/10.1145/70609.70615).
- Victor, Bret (2013). "Media for thinking the unthinkable." In: *Presented at the MIT Media Lab on April 4, 2013*.
- Vinh, Khoi (2015). *Design Tools Survey | The Tools Designers Are Using Today*. <http://tools.subtraction.com/index.html>. (Visited on 08/17/2018).
- Virzi, Robert A., Jeffrey L. Sokolov, and Demetrios Karis (1996). "Usability Problem Identification Using Both Low- and High-fidelity Prototypes." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '96. New York, NY, USA: ACM, pp. 236–243. ISBN: 0-89791-777-4. DOI: [10.1145/238386.238516](https://doi.org/10.1145/238386.238516).
- Vistisen, Peter (2016). *Sketching with animation: using animation to portray fictional realities ? aimed at becoming Factual*. 1st ed. Aalborg Universitetsforlag.
- Vistisen, Peter and Soren Bolvig Poulsen (2017). "Return of the Vision Video: Can corporate vision videos serve as setting for participation?" In: *Nordic Design Research (NORDES)*. 7. Nordes Digital Archive.
- Walker, John A. and Judy Attfield (1989). *Design history and the history of design*. Vol. 79. Pluto press London.
- Warner, Jeremy and Philip J. Guo (2017). "CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers." In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. New York, New York, USA: ACM Press, pp. 1136–1141. ISBN: 9781450346559. DOI: [10.1145/3025453.3025876](https://doi.org/10.1145/3025453.3025876).

- Wenger, Etienne (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge university press.
- Wiemann, Meike (2016). "Patterns as a tool for collaboration: A case study of collaboration between designers and developers through user interface pattern libraries." eng. PhD thesis. Universitet Umeå.
- Wong, Richmond Y. and Deirdre K. Mulligan (2016). "When a Product Is Still Fictional: Anticipating and Speculating Futures through Concept Videos." In: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems - DIS '16*. New York, New York, USA: ACM Press, pp. 121–133. ISBN: 9781450340311. DOI: [10.1145/2901790.2901801](https://doi.org/10.1145/2901790.2901801).
- Wong, Yin Yin (1992). "Rough and Ready Prototypes: Lessons from Graphic Design." In: *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems*. CHI '92. New York, NY, USA: ACM, pp. 83–84. DOI: [10.1145/1125021.1125094](https://doi.org/10.1145/1125021.1125094).
- Young, Emilie and Russell Greenlee (1992). "Participatory video prototyping." In: *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems - CHI '92*. New York, New York, USA: ACM Press, p. 28. DOI: [10.1145/1125021.1125047](https://doi.org/10.1145/1125021.1125047).
- Zimmerman, John (2005). "Video Sketches: Exploring Pervasive Computing Interaction Designs." In: *IEEE Pervasive Computing* 4.4, pp. 91–94. ISSN: 1536-1268. DOI: [10.1109/MPRV.2005.91](https://doi.org/10.1109/MPRV.2005.91).

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both \LaTeX and \LyX :

<https://bitbucket.org/amiede/classicthesis/>

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

<http://postcards.miede.de/>

Thank you very much for your feedback and contribution.

Final Version as of May 2, 2019 (version 3.0).

Titre : Prototypage interactif d'interactions: des prototypes jetables au prototypage recyclable

Mots clés : Interaction homme-ordinateur, design interactif, prototypage rapide, systèmes informatique, vidéo

Résumé :

Je propose de créer des outils de prototypage rapide qui puissent efficacement supporter la création d'artéfacts à la fois jetables et réutilisables pour esquisser de nouvelles interactions dans les premières phases du processus de design.

Les designers font face à deux écueils majeurs à l'utilisation de la vidéo en design d'interaction : le temps nécessaire pour filmer et celui nécessaire pour éditer. J'ai développé VIDEOCLIPPER pour aider le processus de création de vidéo. Je présente les résultats de trois études sur le terrain avec des étudiants en design d'interaction. Les résultats suggèrent que les participants passent moins de temps à capturer et éditer avec VIDEOCLIPPER qu'avec les autres outils vidéos. J'ai ensuite créé MONTAGE, un outil de prototypage vidéo qui permet aux designers de progressivement augmenter leurs prototypes papier avec des maquettes numériques pour faciliter la création, la réutilisation et l'exploration d'interactions dynamiques. Je décris comment MONTAGE améliore le prototypage vidéo en combinant la vidéo avec des

maquettes numériques animées et encourage l'exploration d'autres contextes d'utilisation tout en permettant le prototypage de styles d'interaction différents. Les designers et développeurs professionnels ont souvent du mal à effectuer la transition de la représentation du design à son implémentation concrète. Nous avons découvert que les pratiques actuelles entraînent des redondances entre le travail des designers et celui des développeurs et des divergences entre le design et son implémentation. Nous identifions trois types de problèmes.

Je propose quatre principes de design pour créer des outils qui limitent ces problèmes. Ces principes sont utilisés pour créer ENACT, un environnement interactif de prototypage d'interactions tactiles. Les résultats de deux études suggèrent que ENACT aide les participants à détecter plus de cas extrêmes, augmente la participation des designers et offre de nouvelles possibilités de co-création.

Les outils présentés mettent en œuvre une approche du prototypage que je nomme "Takeaway Prototyping" ou prototypage recyclable.

Title : Interactive prototyping of interactions: from throwaway prototypes to takeaway prototyping

Keywords : Human-computer interaction, interaction design, rapid prototyping, computer systems, video

Abstract : I argue that rapid prototyping tools can effectively support reusable as well as throwaway artifacts for sketching interaction in early-stage design.

Designers experience two main barriers to use video in interaction design : the time to capture and edit the video artifacts. I created VIDEOCLIPPER, a tool that embodies an integrated iterative design method that rewards discipline but permits flexibility for video prototyping. I present field studies with interaction design students using VIDEOCLIPPER in three design courses. Results suggest that participants spend less time capturing and editing in VIDEOCLIPPER than with other video tools.

I created MONTAGE, a prototyping tool for video prototyping that lets designers progressively augment paper prototypes with digital sketches, facilitating the creation, reuse and exploration of dynamic interactions. I describe how MONTAGE enhances video prototyping by combining video with digital animated sketches, encourages the exploration of different contexts of use, and supports prototyping of different interaction styles.

I investigate how early designs start being implemented into interactive prototypes. Professional designers and developers often struggle when transitioning from the illustration of the design to the actual implementation of the system. We find that current practices induce unnecessary rework and cause discrepancies between design and implementation and we identify three recurring types of breakdowns.

I propose four design principles to create tools that mitigate these problems : *Provide multiple viewpoints, maintain a single source of truth, reveal the invisible and support design by enactment.* We apply these principles to create ENACT, an interactive live environment for prototyping touch-based interactions. We introduce two studies to assess ENACT and to compare designer-developer collaboration with ENACT versus current tools. Results suggest that ENACT helps participants detect more edge cases, increases designers' participation and provides new opportunities for co-creation.

The presented tools outline a new prototyping approach that I call Takeaway Prototyping.

