



FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

ProCeeD - A Framework for Continuous Prototyping

Lukas Alfons Alperowitz



FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit 1
Angewandte Softwaretechnik

ProCeeD - A Framework for Continuous Prototyping

Lukas Alfons Alperowitz

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Hans Michael Gerndt

Prüfer der Dissertation: 1. Prof. Bernd Brügge, Ph.D.
2. Prof. Dr. Horst Lichter

Die Dissertation wurde am 04.07.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.09.2017 angenommen.

Prototypes are used in software engineering to close the gap between an abstract idea and the actual software solution. Prototypes facilitate a shared mental model between the stakeholders of a software project. Several techniques have been proposed to develop and deliver prototypes, but the continuous delivery of early prototypes to stakeholders during requirements elicitation has not yet been addressed. This is an obstacle especially in projects with unclear and vague requirements where early feedback from users is of great importance.

This dissertation introduces the ProCeeD Framework and the Prototyper tool for delivering prototypes and obtaining feedback from stakeholders. ProCeeD allows the exploration of requirements by automating the delivery of these prototypes and provides a common feedback process for revolutionary as well as evolutionary prototypes. Prototyper is a software delivery tool based on ProCeeD and allows the delivery of executable prototypes to the target environment. Prototyper has been realized for the domain of mobile application engineering and has been successfully applied in projects in both academia and industry.

ProCeeD and Prototyper have been empirically evaluated in four case studies involving 25 projects with more than 200 developers. The results show that teams who adopt ProCeeD reduce the duration of a prototyping iteration compared to teams not adopting the framework. In addition, projects using ProCeeD exhibited a 2-fold increase in prototype deliveries to the target environment compared to projects not applying the framework.

Prototypen in der Softwareentwicklung schließen die Lücke zwischen einer abstrakten Idee und ihrer Umsetzung. Prototypen dienen als Diskussionsgrundlage und ermöglichen Entwicklern und Stakeholdern ein gemeinsames mentales Modell zu entwickeln. Während in der Vergangenheit mehrere Techniken zur Erstellung von Prototypen vorgeschlagen worden sind, wurde die Rolle der Lieferung von frühen Prototypen an Stakeholder bislang nicht adressiert. Dies ist gerade in Projekten mit unklaren und vagen Anforderungen, bei der die frühzeitige und regelmäßige Diskussion mit den Stakeholdern notwendig ist, von Nachteil.

In dieser Dissertation beschreiben wir das ProCeeD Framework und das darauf aufbauende Prototyper Tool. ProCeeD erlaubt die Lieferung von revolutionären sowie evolutionären Prototypen in einem automatisierten und wiederholbaren Prozess. Des Weiteren definiert ProCeeD einen einheitlichen Feedback-Prozess für revolutionäre sowie evolutionäre Prototypen, der die Kommunikation zwischen Entwicklern und Stakeholdern bereits in der Phase der Anforderungsermittlung unterstützt und unabhängig von der Art des Prototypen genutzt werden kann. Mit dem Prototyper Tool stellen wir eine Konzeptimplementierung von ProCeeD für die Lieferung von ausführbaren Prototypen vor. Prototyper wurde mit einem Fokus auf mobile Anwendungen implementiert und erfolgreich im universitären Umfeld und in der Industrie angewendet. ProCeeD und Prototyper wurden empirisch in vier Fallstudien mit 25 Projekten und mit mehr als 200 Entwicklern evaluiert. Die Ergebnisse zeigen, dass Teams, die ProCeeD nutzen, die Dauer einer Prototyping-Iteration im Vergleich zu Teams, die das Framework nicht anwenden, deutlich reduzieren. Darüber hinaus zeigten ProCeeD-Teams eine Verdopplung der Lieferungen in die Zielumgebung im Vergleich zu Teams, die ProCeeD nicht anwenden.

Acknowledgements

The last years at TUM were some of the best of my life, especially due to the people I met and had the pleasure to work with. First of all Bernd Brügge: Thank you for creating this unique environment and all the possibilities. You taught me how to deal with uncertainty, to catch opportunities and grow personally on the way. Also thank you to Horst Lichter for enthusiastically being the second advisor of this research.

A special thanks to the iPraktikum team! Dora Dzvonyar for the great collaboration, we spent months together and every time we pushed the boundaries a bit further. Not to forget all the project teams I supervised, it was your enthusiasm and motivation which brought things forward. Thanks to Irina Camilleri, Michael Fröhlich and Marius Schulz for your great work during the courses! But also special thanks to the media team Ruth Demmel, Andreas Jung, Martin Meir, Felix Neumann and Ernst Graf. You invested way more than anyone could expect.

Thanks to everyone in the ProCeeD project who made this dissertation a reality. Stefan Kofler for your professional work on Prototyper, it was a pleasure to work with you. Marie Weintraud, Julian Geistbeck, Paul Schmiedmayer and Lara Marie Reimer, you all made great contributions to this project!

I am grateful to have worked with a group of great colleagues at the chair. Constantin Scheuermann for your encouragement and long bicycle rides in the evening, Jan Ole Johanßen for the discussions and ideas, Andreas Seitz for the positive energy, Stephan Krusche who introduced me to the chair and all others. Thank you for the good times, the great discussions and your feedback. We not only shared offices, we spent years of our life together.

Many thanks to Helma Schneider and the whole infrastructure group. We may have worked many long evenings but when we look back and see what has changed in the last three years, it was worth the effort! Not to forget Monika Markl and Uta Weber, thank you for pulling the strings in the background and making it all work out.

A big thank you to my family, for all the support and advice. There is nobody who knows me better. Finally and most importantly thank you Karo. We met on the way and without your support, patience and love nothing of this would have been possible.

List of Figures	xiii
List of Tables	xv
1 Introduction	1
2 Foundations	5
2.1 Requirements Engineering	5
2.1.1 Stakeholders	6
2.1.2 User Involvement	7
2.1.3 Requirements Elicitation	8
2.2 Prototyping	8
2.2.1 Prototyping as a Process	9
2.2.2 Prototypes as Artifacts	12
2.2.3 Tool Support	14
2.2.4 Market Overview	15
2.3 Continuous Delivery	16
2.4 Continuous Prototyping	18
3 ProCeeD Framework	21
3.1 Scenarios	21
3.1.1 Deliver Prototype	22
3.1.2 Receive Prototype	24
3.1.3 Deliver Multiple Prototypes	25
3.1.4 Deliver Hybrid Prototype	27
3.1.5 Decide on a Release Schedule	29
3.1.6 Analyze the Release Process	30
3.2 Use Cases	31
3.3 Requirements	33
3.3.1 Functional Requirements	33
3.3.2 Nonfunctional Requirements	35
3.4 Object Model	36

3.5	Dynamic Model	38
3.5.1	Release	38
3.5.2	Workflows	39
3.6	Subsystem Decomposition	42
3.7	Hardware/Software Mapping	45
4	Prototyper	47
4.1	User Interface	47
4.2	Hardware/Software Mapping	55
5	Evaluation	61
5.1	Overview	61
5.1.1	Hypotheses	62
5.1.2	Methodology	64
5.2	Case Study I: University Capstone Course	65
5.2.1	Design	65
5.2.2	Quantitative Results	70
5.2.3	Qualitative Results	74
5.2.4	Discussion	78
5.3	Case Study II: Process Metrics	81
5.3.1	Design	81
5.3.2	Results	84
5.3.3	Discussion	84
5.4	Case Study III: Industry	86
5.4.1	Design	86
5.4.2	Results	86
5.4.3	Discussion	89
5.5	Case Study IV: Storyboard-based Requirements Elicitation	90
5.5.1	Design	90
5.5.2	Results	91
5.5.3	Discussion	92
5.6	Summary	93
5.7	Threats to Validity	93
6	Conclusion	95
6.1	Contributions	95
6.2	Future Work	97
	Bibliography	99

1.1	ProCeeD Framework and workflows.	2
2.1	Prototyping as a process - taxonomy (UML Class Diagram).	9
2.2	Prototypes as artifacts - taxonomy (UML Class Diagram).	12
2.3	Paper prototype digitized using the prototyping tool MarvelApp.	16
2.4	Example of a Continuous Delivery pipeline (adapted from [HF10]).	17
2.5	Continuous Prototyping approach.	18
2.6	Prototypes in Continuous Prototyping - taxonomy (UML Class Diagram).	19
3.1	Release plan for scenario S1 and S2.	22
3.2	Release plan for scenario S3.	25
3.3	Release plan for scenario S4.	27
3.4	ProCeeD's requirements exploration use case model (UML Use Case Diagram).	31
3.5	ProCeeD's delivery automation use case model (UML Use Case Diagram).	32
3.6	ProCeeD's process metrics use case model (UML Use Case Diagram).	32
3.7	ProCeeD's object model (UML Class Diagram).	36
3.8	ProCeeD's release model (UML State Machine Diagram).	38
3.9	ProCeeD's workflows.	39
3.10	ProCeeD's requirements exploration workflow (UML Activity Diagram).	39
3.11	ProCeeD's delivery automation workflow (UML Activity Diagram).	40
3.12	ProCeeD's process metrics workflow (UML Activity Diagram).	41
3.13	ProCeeD's process metrics - example	41
3.14	ProCeeD's subsystem decomposition - objects and packages (UML Class Diagram).	43
3.15	ProCeeD's subsystem decomposition - subsystems (UML Component Diagram).	44
3.16	ProCeeD's hardware/software mapping - example (UML Deployment Diagram).	45
4.1	Prototyper's dashboard showing available applications.	47
4.2	ProCeeD components used for release mgmt. (UML Component Diagram).	48
4.3	Prototyper's release management user interface showing revolutionary prototypes (orange) and evolutionary prototypes (green).	49
4.4	ProCeeD components used for release promotion (UML Component Diagram).	49
4.5	Prototyper's release promotion user interface.	50
4.6	Prototyper's release notes user interface.	51

4.7	ProCeeD components used for feedback mgmt. (UML Component Diagram). . .	51
4.8	Prototyper's feedback management user interface.	52
4.9	Prototyper's FeedbackCollector component embedded in a mobile application.	53
4.10	ProCeeD components used for process metrics (UML Component Diagram). .	54
4.11	Prototyper's process metrics user interface showing metrics for a release. . . .	54
4.12	Prototyper's hardware/software mapping (UML Deployment Diagram). Components implemented as part of Prototyper are drawn blue, components based on COTS software are drawn yellow.	56
4.13	Revolutionary and evolutionary prototypes combined into a hybrid prototype.	57
4.14	Mobile app created using Prototyper. Yellow parts are based on a revolutionary prototype, orange parts are implemented in code.	58
5.1	Mapping of the case studies to the ProCeeD workflows.	64
5.2	Case study I: Organizational chart of the course (adapted from [BKA15]). . . .	66
5.3	Case study I: Continuous Prototyping process applied.	67
5.4	Case study I: Tracked events (capital letters) and calculated metrics (arrows). .	68
5.5	Case study I: Amount of prototypes delivered to the client.	70
5.6	Case study I: Amount of prototypes delivered to clients in an automated process.	71
5.7	Case study I: Delivery times revolutionary prototypes.	72
5.8	Case study I: Delivery times evolutionary prototypes.	72
5.9	Case study I: Feedback times revolutionary prototypes.	73
5.10	Case study I: Feedback times evolutionary prototypes.	73
5.11	Case study I: Feedback channels.	73
5.12	Case study I: Prototyper usage (single choice).	74
5.13	Case study I: Statements delivery automation 1.1 - 1.3 (5-likert).	75
5.14	Case study I: Statements delivery automation 2.1 - 2.2 (5-likert).	75
5.15	Case study I: Statements requirements exploration 1.1 - 1.3 (5-likert).	76
5.16	Case study I: Statements requirements exploration 2.1 - 2.3 (5-likert).	77
5.17	Case study I: Statements integration with developer tools 1.1 - 1.3 (5-likert). .	77
5.18	Case study I: Statements integration with developer tools 2.1 - 2.3 (5-likert). .	78
5.19	Case study II: Development workflow (adapted from [BKA15])	81
5.20	Case study II: Metrics Continuous Integration.	82
5.21	Case study II: Metrics Continuous Prototyping.	83
5.22	Case study III: Release plan for DiaApp - example.	87
5.23	Case study III: Four prototyping iterations of the DiaApp dashboard.	88
5.24	Case study IV: Visual backlog creation process (adapted from [ASF17]).	90
5.25	Case study IV: Top-level architecture (example).	91
5.26	Case study IV: Visual backlog showing the system after five iterations.	92
6.1	Contributions.	95

2.1 Prototyping tools classified by goal, applied techniques and delivery process. . .	15
5.1 Case study I: Participants	65
5.2 Case study I: Questionnaire - overview	69
5.3 Case study III: Releases delivered for DiaApp 2	86
5.4 Evaluation: Summary	94

Finding and fixing a problem in a software system early in the development process has been shown to be significantly less expensive than correcting the problem after delivery [BB01]. Dealing with change and being able to elicit and refine requirements even in late development stages is a success factor for innovative software projects. In the waterfall model [Roy+70] requirements are elicited and analyzed only at the beginning of the software development process. Iterative and incremental lifecycle models like the Rational Unified Process have proposed to model requirements elicitation as a workflow throughout the lifespan of a software system [Jac+99]. To reduce the cost and risks caused by changing requirements throughout the stages of the software lifecycle, agile methods such as Extreme Programming or Scrum let developers work in even shorter cycles [Bec00], [SB02].

Various techniques can be applied to elicit requirements: in personal interviews or questionnaires requirements are elicited with the stakeholders. Model-based approaches apply use cases and scenarios written in natural language to structure the elicitation process [Mai98]. Prototyping allows the elicitation of requirements using early abstractions which show the main characteristics and features of an artifact in development. The term ‘prototype’ originates from the greek words *protos*, which translates to ‘original, primitive’ and *typos* which translate to ‘impression’¹. Literally speaking, a prototype is ‘a first or primitive’ form of a product to be developed. Rapid prototyping describes the fast creation of physical prototypes [TB90]. In software engineering Prototypes are created to close the gap between an abstract idea and the proposed software solution by facilitating a shared mental model [Nor12] between the stakeholders. Research shows that software prototyping has a positive benefit on the success of software projects [HL01] and helps to overcome the challenge that the requirements in a project are often unclear to the stakeholders until they are implemented in a system which can be executed and tested [Car00].

In this dissertation we focus on prototypes to support the development of interactive systems where user interface models are the means of communication between the stakeholders. Prototypes for interactive systems can be created with several techniques such as paper-based prototyping, wireframing, storyboarding or wizard-of-oz prototyping: Paper-based prototypes focus on the static layout of the user interface. Wireframe prototypes model the user interaction with low-fidelity sketches of the user interface elements. Storyboard prototyping demonstrates the interaction of the user with the system by modeling a

¹<http://www.dictionary.com/browse/prototype>

flow of interaction steps using images or videos [And89]. Wizard-of-oz prototyping [Kel84] has been applied to evaluate the user interface of a system while a human controls the not yet developed interaction steps of the system.

In addition to this large spectrum of prototyping techniques, prototypes for interactive software systems are developed with a variety of tools and media. The target environments of interactive systems range from desktop computers, smartphones and tablets to wearable devices and in-car entertainment systems. As a result prototypes are often demonstrated and evaluated in environments which differ from the environment the proposed system will be executed in. This can lead to wrong assumptions regarding the system's requirements by the stakeholders. For example, when a mobile application prototype is evaluated with mouse and keyboard on a desktop computer, the user experiences the interactions in a different way than in a mobile environment.

In this dissertation, we see requirements elicitation as a dialectic process where stakeholders define, refine or dismiss requirements based on the regular delivery of these prototypes to the user and in the target environment of the proposed system. In order to create successful systems, the peculiarities of the target environment like the usage context, input modalities or interaction model need to be taken into account already during requirements elicitation. We believe that evaluating prototypes in the target environment facilitates problem understanding, improves the quality of the requirements and - ultimately - the final system. An important aspect is the delivery of these early prototypes. While approaches like Continuous Delivery [HF10] have been developed to automate the delivery of software to the target environment, the delivery of revolutionary prototypes to explore and refine requirements has not yet been addressed. This is a drawback especially in innovation projects with unclear and vague requirements where early and regular feedback from users is of great importance.

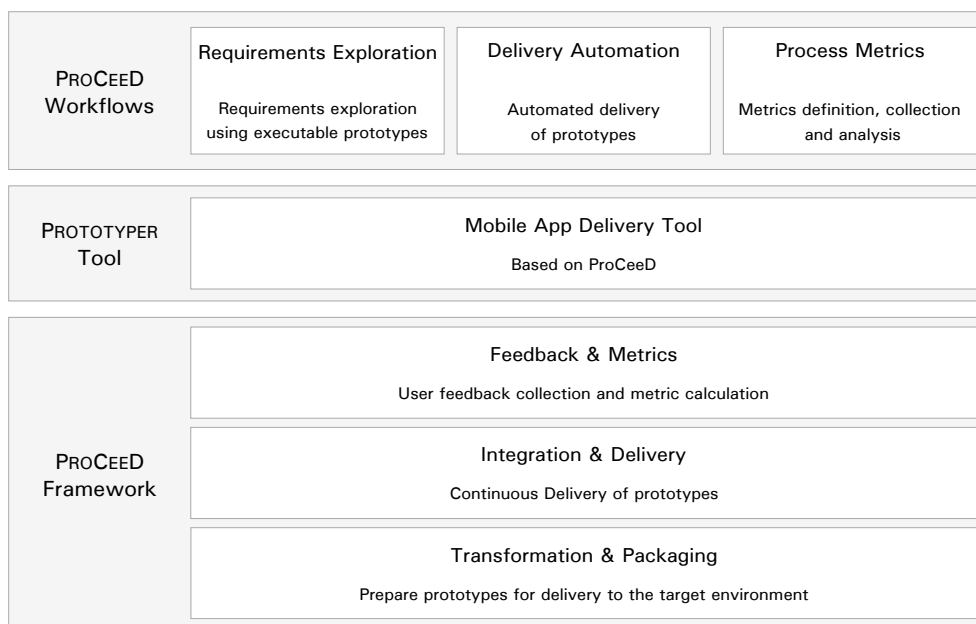


Figure 1.1: ProCeed Framework and workflows.

This dissertation addresses the above-described gap by introducing the ProCeeD Framework for the delivery and feedback management of prototypes and the Prototyper tool which adopts ProCeeD in the field of mobile application engineering. An overview of ProCeeD's architecture and workflows is shown in Figure 1.1. ProCeeD's requirements exploration workflow enables developers to elicit and explore requirements by delivering one or multiple executable prototypes of the proposed system to groups of users. The delivery automation workflow allows developers to deliver prototypes created with a variety of techniques to the target environment using an automated and repeatable process. Stakeholders can execute and evaluate the prototypes and give feedback. The process metrics workflow enables development teams to measure, evaluate and improve their delivery process based on process metrics calculated by ProCeeD. To evaluate ProCeeD we implemented the Prototyper tool in the domain of mobile application engineering. We evaluated ProCeeD and Prototyper in an academic and industrial setting in four case studies. The remainder of this dissertation is structured as follows:

Chapter 2 introduces the terminology used in this dissertation as well as the foundations in the fields of requirements engineering, prototyping and software delivery. It also shows the results of a market study in which we analyzed prototyping tools with a focus on their delivery workflow.

Chapter 3 describes the scenarios and use cases of the ProCeeD Framework and elicits its functional and nonfunctional requirements. It then explains ProCeeD workflows requirement exploration, delivery automation and process metrics. Finally it presents the analysis and system design models of ProCeeD.

Chapter 4 shows the Prototyper application delivery tool which is based on the ProCeeD Framework. It maps Prototyper's user interface to the ProCeeD architecture and presents a detailed mapping of ProCeeD's and Prototyper's components to hardware and software nodes as well as protocols used in the implementation.

Chapter 5 presents the empirical evaluation of ProCeeD in four case studies. The first case study evaluates Prototyper as well as ProCeeD's delivery automation and requirements exploration workflows in a multi-project capstone course with more than 200 developers working in 22 projects. The second case study evaluates ProCeeD's process metrics workflow in a project course across multiple semesters. The third case study shows the application of Prototyper and ProCeeD in a commercial project with a partner from the pharmaceutical industry. The fourth case study describes the use of ProCeeD in a project following a storyboard-based requirements elicitation approach.

Chapter 6 summarizes the contribution of this dissertation and provides an outlook on future work with regard to ProCeeD and the Prototyper tool.

This chapter presents the foundations in the fields of requirements engineering, software prototyping and software delivery and defines the notion of Continuous Prototyping. The chapter is structured as follows: Section 2.1 presents an overview of the field of requirements engineering, in particular its origins and current research topics. It then describes the roles and responsibilities of the different persons involved during requirements engineering. Section 2.2 covers prototyping as a requirements elicitation technique and presents prototyping as a process with different goals, techniques applied and delivery methods used. It then describes prototypes as artifacts and presents a taxonomy to classify prototypes by their fidelity, integration and executability. In addition it presents an overview of prototyping approaches and tools created in science and industry. Section 2.3 introduces the concept of Continuous Delivery. Finally Section 2.4 coins the notion of Continuous Prototyping which integrates prototyping as a requirements elicitation technique with the state of the art in continuous software delivery.

2.1 Requirements Engineering

The first attempts to structure requirements analysis in software projects emerged in the 1970's when DeMarco introduced the structured analysis to model systems functionality and data flows [DeM79]. DeMarco's approach distinguished between the analysis of the current state of a software system and a model of the desired state, an early form of requirements specifications. McMenamin and Palmer later proposed Essential System Analysis and distinguished between the *essence* and *incarnation* of a software system, a distinction DeMarco's approach was lacking. While the *essence* describes the characteristics of a system in a perfect world and without any technological constraints, the *incarnation* of a system considers the actual implementation constraints given by technology or people [MP84]. By applying such an approach, requirements analysts could now distinguish between the problem to be solved and a feasible technical solution.

Until the 1970's, requirements were mainly written by developers and engineers. Software systems were mostly created to automate existing processes. Users of such systems were usually engineers or programmers and the system's requirements could be stated upfront and formally verified after their implementation. The focus of requirements engineering methods was on the definition of unambiguous and complete requirements and did not involve actual users of the proposed system. Dijkstra summarized the situation as

follows: "[T]he notion of 'user' cannot be precisely defined, and therefore has no place in computer science or software engineering." [Dij79].

After IBM presented the personal computer in 1981 [IBM81], the rise of interactive software systems and their users facilitated a paradigm shift in requirements engineering: Eliciting and analyzing requirements without involving the users of a system was no longer enough. With the Spiral Model [Boe88], Boehm claimed to reduce the risk of large software projects by employing an iterative and prototype-based design approach where requirements could be altered between iterations. Subsequent iterative and incremental lifecycle models like the Rational Unified Process [Jac+99] modeled requirements engineering as a workflow throughout the lifespan of a software system. To further reduce the risk caused by changing requirements throughout the software lifecycle, agile methods such as Extreme Programming and Scrum let stakeholders review and refine the requirements in short development cycles [Bec00], [SB02]. Today, requirements engineering is concerned with the elicitation, documentation, negotiation, validation and management of requirements throughout the whole software lifecycle [Poh10].

In the following sections, we present an overview of the different roles involved in requirements engineering and discuss techniques to elicit requirements.

2.1.1 Stakeholders

The term *stakeholder* has its origin in the area of strategic management [Fre10]. In software engineering, especially in requirements engineering, different definitions for the notion of a stakeholder exist. Macaulay was the first to define stakeholders in the field of requirements engineering as: "all those who have a stake in the change being considered, those who stand to gain from it, and those who stand to lose." [Mac93]. Kotonya and Sommerville saw stakeholders as "people or organizations who will be affected by the system and who have a direct or indirect influence on the system requirements." [KS98], similar to Conger who stated stakeholders are "The people and organizations affected by the application" [Con93]. For this dissertation, we define stakeholders according to Glinz and Wieringa as "a person or organization who influences a system's requirements or who is impacted by that system." [GW07].

We distinguish the following stakeholders: requirements analysts, developers, interaction designers, clients, users and promoters. *Requirements analysts* are concerned with the exploration, elicitation and analysis of the system's requirements. Using techniques described in Section 2.2 requirements analysts collaborate with other stakeholders to define and refine requirements.

Developers contribute artifacts like source code or architectural concepts. Depending on the project organization developers fulfill additional roles like managing delivery or code review activities.

Interaction Designers are especially involved during the ideation and development of interactive software systems. Their responsibilities are the definition, evaluation and refinement of the user interface and user interaction model of the proposed system.

Clients are the persons or the company who pay, contract or request a software application. A client can have varying domain knowledge and decision power [BD09]. When the client is a domain expert who is very familiar with the problem to be solved, he can give valuable feedback to the developers [BPKR09].

Users are the persons who will use the proposed system [BD09]. They can provide explicit and implicit feedback to other stakeholders. Explicit feedback is actively provided in the form of textual or spoken comments on the user interaction design of a prototype. Implicit feedback is any feedback which can be collected without the active participation of the user, e.g. through the collection of the interaction steps a user took while using a software application.

Witte and Hausschild introduced the notion of *promoters*. A promoter is an invisible stakeholder who may not influence the requirements of a software system, but still has an impact on its success. While promoters stay mostly invisible during development, they have a viable interest in the development of a software system. By using their expert knowledge or position in the company they help to overcome organizational barriers and obstacles [Wit73], [Hau98].

2.1.2 User Involvement

Methods to involve users during software development have been researched since the 1980's, when the concept of user-centered system design [ND86] appeared. Gould and Lewis proposed three principles for user centered design [GL85]: early focus on users, empirical measurement using prototypes and iterative design. They recommended that potential users of the system become part of the design team from the beginning. As not each potential user of a system can be actively involved during requirements engineering, several concepts emerged to allow the creation of user models: Jacobson introduced the term *actor* as a superclass for the different users of a system. He defined an actor as everyone who "interacts with the system" and "needs to exchange information with the system" [Jac93].

Cooper coined the term *persona*. A persona is a fictional representation of a group of users [Coo99]. Personas are created to allow developers and designers to model the different kinds of users which will use the proposed system later on.

Nowadays the involvement of actual users is a desired practice during software development [Kuj03], [Pag13]. However, Kujala found that "early user involvement is still a rare phenomenon in ordinary development projects" [KKLK05]. Moreover, Kujala sees "early user involvement [...] to be a powerful way of improving the requirements quality and project success" [KKLK05].

2.1.3 Requirements Elicitation

Requirements elicitation is the extraction and creation of the requirements of a system by its stakeholders [Poh10]. An important aspect of requirements elicitation is the scoping of the problem that needs to be solved by defining the system's boundaries and identifying its stakeholders [KS98]. Requirements elicitation is a challenging activity because stakeholders who know about the problem domain may not understand the technical aspects of the system under development. On the other hand developers, who know about software engineering processes, are sometimes not familiar with the problem domain [HB95],[KS98]. Many techniques can be applied to elicit requirements:

- *People-oriented techniques* include the use of questionnaires and the conduction of interviews with clients or users. In addition, the analysis of existing documentation is used in reengineering projects.
- *Creative techniques* elicit requirements using brainstorming or focus groups with multiple stakeholders [PEM03].
- *Model-based* approaches structure the requirements elicitation process by applying a set of goals or scenarios defining the type and detail of the information to be gathered. A well-known goal-based elicitation approach is KAOS [LDL98], which proposes to derive a systems requirements from a set of goals. Examples for requirements elicitation approaches involving scenarios are CREWS [Mai98] and SCRIPT [Sta+12].
- *Cognitive techniques* use protocol analysis where subjects execute a certain task while thinking aloud. Another technique is card sorting where subjects are asked to structure domain entities by grouping cards containing possible entities [NE00].

Prototyping is a requirements elicitation technique that emerged in the 1970's. As prototyping is the focus of this dissertation, we describe the prototyping process and prototypes as artifacts in detail in the next section.

2.2 Prototyping

The term *Prototype* itself originates from the greek words *protos*, which translates to 'original, primitive' and *typos* which translate to 'impression'¹. Literally, a prototype is 'a first or primitive' form of a product to be developed. In industrial production, a prototype represents an early sample showing the main characteristics and features of the product under development. In this context, the term 'Rapid Prototyping' describes the fast creation of physical prototypes [CLL03].

Bruegge and Dutoit refer to software prototypes as simplified versions of a system [BD09], similarly to Guida et al. who state that "[a] prototype is a dynamic model of the software system under construction" [GLZ99]. Prototypes demonstrate parts of a proposed

¹<http://www.dictionary.com/browse/prototype>

system in a simplified manner and can be used to present the vision of a system [BBLZ96]. Prototyping can be understood as "the construction of an executable system model to enhance understanding of the problem and identify appropriate and feasible external behaviors for possible solutions" [HDK93]. A prototype demonstrates the feasibility of the functionality of a system to identify possible risks; it provides a means of communication between developers and users and it is used to explore and elicit requirements which are not completely known upfront [Rup+07], [Urb92], [BKKZ92].

Prototyping techniques have been part of software engineering lifecycle models since the 1980's. For example, Boehm incorporated the creation of prototypes in his spiral model [Boe88]: he required developers to create prototypes in each iteration to reduce the project risk. Jacobson proposed the creation of prototypes in the *Unified Process* to evaluate architectural decisions as well as to demonstrate the mitigation of technical risks [Jac+99], [Kru04]. Beck proposes to create User Stories to describe requirements in *Extreme Programming* [Bec00]. A developer creates a prototype for each user story to demonstrate the requirement to the customer or users. In the following, we analyze prototyping by the underlying process and the resulting artifacts.

2.2.1 Prototyping as a Process

Prototype development can be modeled as process with a certain goal. Prototypes can be created using different techniques and are delivered to project stakeholders in a manual or automated delivery process. Figure 2.1 depicts this taxonomy.

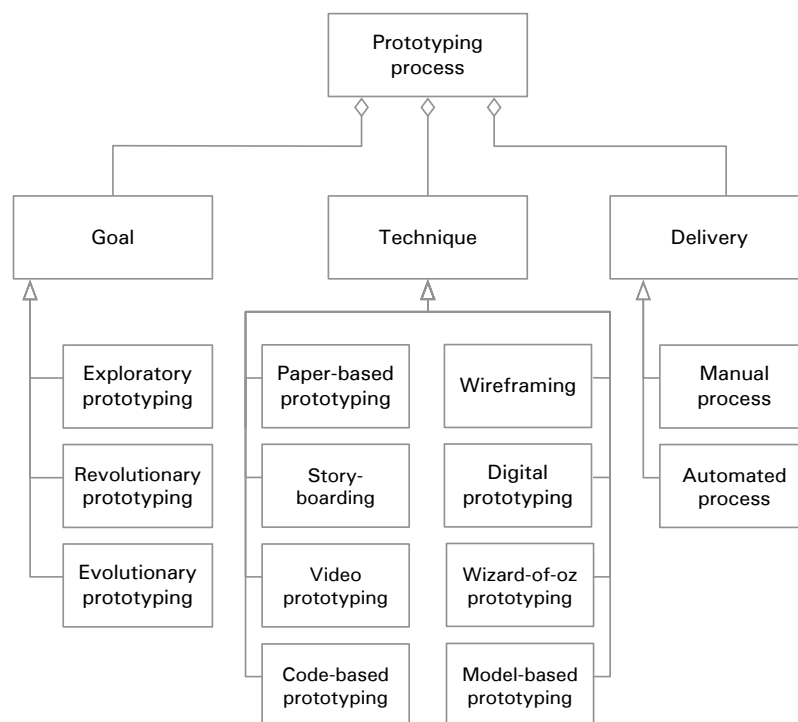


Figure 2.1: Prototyping as a process - taxonomy (UML Class Diagram).

Goal

Graham distinguishes three classes of prototyping processes, each of them fulfilling a different goal [Gra94b]. *Exploratory Prototyping* is applied when the problem to be solved is not yet defined, e.g. to envision new ideas or as an approach for unsolved problems [Gra94b]. *Revolutionary Prototyping*, also known as ‘throw-away prototyping’, has the goal to elicit requirements for a system by discussing possible alternative solutions and to facilitate communication with the stakeholders, e.g. developers, clients or users [Gra94b]. An important aspect of revolutionary prototyping is that the created prototypes are used to discuss ideas and possible solutions and are ‘thrown away’ afterwards. Creating multiple revolutionary prototypes can therefore introduce overhead into the software development process. Appropriate tools which keep the effort for the creation of revolutionary prototypes low are required. Furthermore, exposing a revolutionary prototype to e.g. a client or user can lead to disappointment when it becomes clear that the final system may not implement all features shown in the prototype [KHSI12].

Evolutionary Prototyping describes an incremental and iterative prototyping approach. Evolutionary prototypes are created and refined in cycles and evolved towards the actual system. An evolutionary prototype is implemented and delivered using the same tools and programming language as the actual system [Gra94b], [Flo84].

Davis proposed the concept of *Operational Prototyping* by combining evolutionary and throw-away prototyping. Operational prototyping asks developers to create a throw-away copy of the currently developed evolutionary prototype. An experienced developer alters the prototype together with the client and according to their ideas. When the client accepts the prototype, the change is integrated into the evolutionary prototype [Dav92].

Technique

Prototypes can be created with several techniques such as paper-based prototyping, wire-framing, storyboarding, digital prototyping, video prototyping or wizard-of-oz prototyping. These are shown in Figure 2.1 as an overview:

Paper-based prototyping describes the sketching of a user interface on paper [Sny03]. The sketches are then used to evaluate the user interface. For such an experiment a facilitator puts the prototype on a table and narrates the interaction steps the user should take. The user interacts with the paper prototype as he would with the real system. Paper prototypes allow users to participate in the prototyping process by expressing ideas and providing feedback. According to Snyder, paper prototypes allow the fast collection of user feedback while requiring little effort and expertise to create [Sny03]. Developers are likely to accept changes as they are less emotionally attached to the prototype [DKA14]. However, creating and especially modifying a paper prototype can be a time-consuming activity.

Wireframing has its origin in 3D computer graphics, where it describes the visual representation of an object by only showing its outer structure without any texture or color. In software engineering, a wireframe prototype is a low-fidelity drawing of the user interface showing only the rough position and size of its elements. A wireframe does not include any detailed design like colored images. Multiple wireframe prototypes can be used to prototype the user interface of a system [Sny03]. Wireframe prototypes can be created using software tools and are then distributed electronically. Sefelin et al. found that prototypes drawn on paper result in the same quantity and quality of critiques and suggestions from users as wireframes created with software [STG03]. Similarly, Walker et al. found paper and wireframe prototypes to be equally suitable for usability testing [WTL02].

Storyboarding is a well-established practice in the filming industry to sketch out movies before filming them. Movie storyboards consist of sequences of frames showing a specific drawing representing the movie shots. Applied to software prototyping, the frames show e.g. an image of the user interface or a narrated part explaining the users interactions with the system [And89].

Digital prototyping denotes the creation of prototypes with software tools. Jørgensen et al. developed TAP [JCK10], a tool to create prototypes for mobile devices. Lasecki et al. used crowdsourcing techniques to create digital prototypes with multiple persons at the same time [Las+15].

Video prototyping is a scenario-based technique to model the user interface and interaction model of interactive systems. Vertelney describes an approach to demonstrate a user interface with a small implementation effort as only the parts of the interface which are shown in the video need to be designed and implemented [Ver89]: The video prototype shows the proposed interface and also the interactions of the user with it. Creighton developed a video-based approach where objects and their relationship can be annotated in scenario movies. Based on the annotations, UML models are generated from the scenario movie [COB06].

Wizard-of-oz prototyping models the user interface of a not yet implemented system with a human carrying out the interaction steps of the proposed system. If a user e.g. clicks a button, a human will fake the system's response on the display without the user noticing [Kel84]. Experiments with wizard-of-oz prototypes need to be executed in a way that the participating users believe the system has already been implemented. Wizard-of-oz prototypes are a cost-efficient way to explore different interaction methods and usability aspects without the need of implementing these alternatives upfront. Ganhor et al. have used executable, low fidelity prototypes to conduct wizard-of-oz experiments with a mobile application [GGSF14]. Peters used wizard-of-oz prototyping to explore requirements for building control system with multiple modalities [Pet16].

Finally, *Model-based prototyping* describes techniques where prototypes are generated automatically from any kind of model, e.g. a formalized use case model. *Code-based prototyping* describes the implementation of a prototype in a programming language.

Delivery

Prototyping can be distinguished by the process applied for delivering a prototype. For this dissertation, we define two possibilities: A prototype is delivered using an *automated process* when the prototype can be delivered to the target environment by a developer with only few interactions. This also includes that a developer can apply the same delivery process for a prototype as he uses for the delivery of the proposed system.

A prototype is delivered using a *manual process* when the prototype is delivered with a method which diverges from the method applied for delivering the proposed system. Manual delivery methods are e.g. email, paper or proprietary solutions of prototyping tool vendors.

2.2.2 Prototypes as Artifacts

Prototypes can be classified as artifacts according to their focus, integration, precision and executability. Figure 2.2 visualizes this taxonomy.

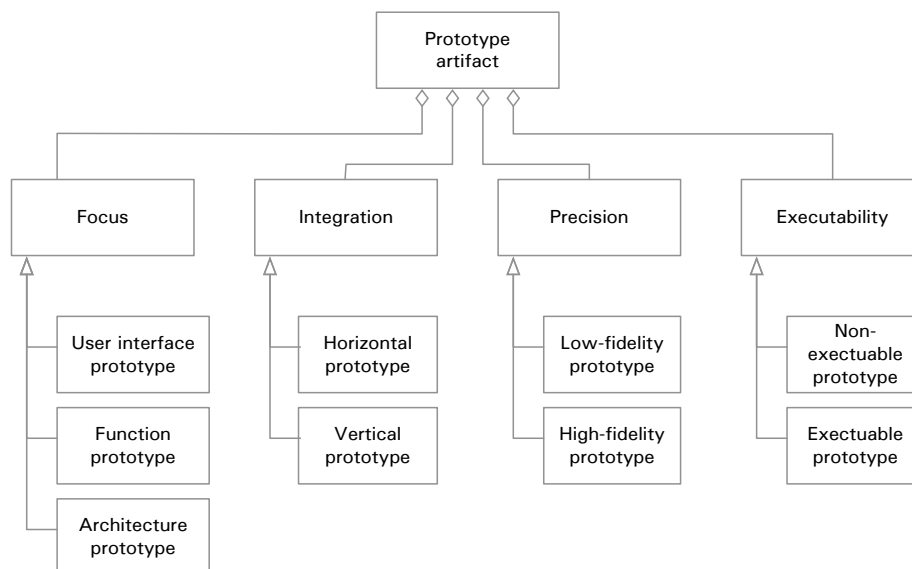


Figure 2.2: Prototypes as artifacts - taxonomy (UML Class Diagram).

Focus

Prototypes can vary in their focus depending on the nature of the proposed system and its corresponding requirements. A *user interface prototype* is created to discuss e.g. the user interaction model or user interface design of an interactive system with its stakeholders. A *function prototype* is used to evaluate technical aspects like an algorithm. When the prototype is created to evaluate aspects of the proposed systems architecture, e.g. the communication between different subsystems, an *architecture prototype* is the appropriate choice.

Integration

Nielsen distinguishes prototypes by their integration into horizontal and vertical prototypes [Nie94]. A *horizontal prototype* covers e.g. the user interface of the actual system, but not its underlying objects and methods. In contrast, a *vertical prototype* implements a slice of the actual system, i.e. parts of the user interface with the corresponding objects and methods. While a horizontal prototype is used to demonstrate the user interaction model, a vertical prototype is usually applied to demonstrate a single piece of functionality [BKKZ92].

Fidelity

Prototypes can be distinguished by their precision, denoting the level of fidelity of the prototype.

Low-fidelity prototypes show a model of the user interface of the actual system. User interface elements are drawn as sketches and may not be precisely aligned. Low-fidelity prototypes are created on paper or with software and are used to iteratively explore the design space. Rudd et al. construct low-fidelity prototypes to depict concepts and design alternatives or user interface designs [RSI96]. They state that low-fidelity prototypes offer limited functionality and interactivity and require a facilitator who knows the application thoroughly to demonstrate the interaction steps of the application [RSI96]. Koehler et al. state that low-fidelity prototypes tend to distract stakeholders less from the actual prototype by avoiding details when compared to high-fidelity prototypes [KHSI12].

In contrast, *high-fidelity* prototypes are created with attention to detail, often using detailed designs. Using high-fidelity prototypes users can interact with prototype as if it were the real system. In general, high-fidelity prototypes trade speed for accuracy [RSI96]. They are not as quick and easy to create as low-fidelity prototypes, but they model the interface and interaction design of the actual system more precisely. Although prototyping tools exist to simplify their creation, high-fidelity prototyping requires substantial effort and can introduce overhead in the development process. Sefelin states that using high-fidelity prototypes, stakeholder feedback often addresses only details in the user interface and that stakeholders are reluctant to challenge the developer as they think this is already the actual system [STG03].

Virzi compared usability problems when using low- and high-fidelity prototypes and found that both approaches find the same set of usability problems along the development process [Vir89]. They further show that low-fidelity prototyping is not only effective during initial design stages, but also throughout the product development lifecycle.

Executability

Prototypes can further be distinguished by their executability. We define a prototype to be *executable* as the prototype can be executed in the target environment of the proposed system and a user can interact with the prototype using the interaction methods of the proposed system. If the proposed system is aiming for a touch-based user interface on a mobile device, then the executable prototype should also be usable with that user interface.

2.2.3 Tool Support

Creating prototypes with varying goals and techniques can be accomplished with commercial and non-commercial prototyping tools. In the following two sections, we give an overview of solutions developed in research and present a market overview of commercially available prototyping tools. For this dissertation we focus on tools and approaches to create user interface prototypes.

In the 1990s, Vlissides and Tang created *IBuild*, a user interface builder which allows the composition of user interfaces independently from the proposed system. With *Ibuild* developers can prototype user interfaces and apply *Ibuild*'s code generation feature to export the interface in the programming language and using the user interface toolkit of the proposed system [VT91]. Landay created *SLICK*, a tool to quickly create interactive, digital sketches of interactive systems. Using *SLICK*, developers can sketch low-fidelity user interfaces using a digital stylus. In a second step, they can connect the drawn interface elements with each other. For example, a developer can connect a button with dialog box. Every time a user clicks the button, the dialog box will appear. User interface prototypes drawn with *SLICK* provide the advantage of being interactive when compared to paper prototypes [Lan96]. Stangl's *SCRIPT* editor allows the creation of scenario-based user interface prototypes. *SCRIPT* allows developers to evolve scenarios and user interface prototypes concurrently. In addition, it helps developers to keep both in a consistent state during development by visualizing elements where user interface prototype and scenario deviate from each other: If a developer modifies a scenario, *SCRIPT* makes sure that the corresponding user interface conforms with the scenario and vice versa [Sta+12]. Hoffmann presented the *OpenUMF tool* which allows developers to automatically transform use case models into user interface prototypes whenever the use case model has changed. To allow this automated code generation, *OpenUMF* implements a formalized notation for denoting use cases [HL13].

2.2.4 Market Overview

Table 2.1 depicts the results of a market survey we conducted in 2016 to assess commercial and non-commercial available prototyping tools.

Table 2.1: Prototyping tools classified by goal, applied techniques and delivery process.

Tool	Website	Technique	Delivery
Balsamiq	balsamiq.com	Wireframing Digital prot.	Manual (PDF)
Moqups	moqups.com	Wireframing Digital prot.	Manual (PDF, HTML)
AppCooker	appcooker.com	Digital prot.	Manual (PDF, Proprietary)
InVision	invisionapp.com	Wireframing Digital prot.	Manual (PDF, Proprietary)
MarvelApp	marvelapp.com	Paper-based prot. Wireframing Digital prot.	Manual (HTML, App)
Keynote	apple.com	Wireframing	Manual
PowerPoint	microsoft.com	Digital prot.	(PDF, Proprietary)
Android Studio	android.com	Code-based prot.	N/A
Xcode	apple.com	Digital prot.	
Proto.io	proto.io	Wireframing Digital prot.	Manual (PDF, Proprietary)
Paintcode	paintcodeapp.com	Digital prot.	N/A
JustInMind	justinmind.com	Wireframing Digital prot.	Manual (HTML, Proprietary)

All tools analyzed allow developers to define a certain amount of interactivity for the prototypes ranging from the definition of transition between different screens up to completely programmable environments where e.g. input and output parameters for user interface elements can be defined. Figure 2.3 shows an example of a paper-based prototype digitized using the prototyping tool MarvelApp². Here a developer has digitized a paper-based prototype in a first step. In a second step, he has linked several elements of the prototype with another screen to make the prototype interactive. Regarding the applied prototyping technique, most of the tools allowed the creation of low-fidelity wireframes and low- and high-fidelity digital prototypes.

²<http://www.marvelapp.com>

IDEs such as Android Studio³ or Apple Xcode⁴ can be used to create prototypes that are code-based or to a certain degree digital. While most of the tools allow the manual delivery of the created prototypes e.g. as PDF documents, some tools allow the delivery of prototypes as executable application. *AppCooker* provides proprietary apps for common ecosystems like iOS and Android to execute prototypes created with the tool. Only one tool, *MarvelApp*, allowed the creation of mobile apps which can be deployed to the target environment. Such an app can be delivered 'as-is' and only be edited using the prototyping tool itself. The delivery process itself is managed by *MarvelApp*.

We conclude that while many tools exist to create revolutionary and evolutionary prototypes for interactive systems, available tools neglect the delivery of these prototypes. If a tool supports the delivery of the created prototypes, it uses a manual and proprietary approach.

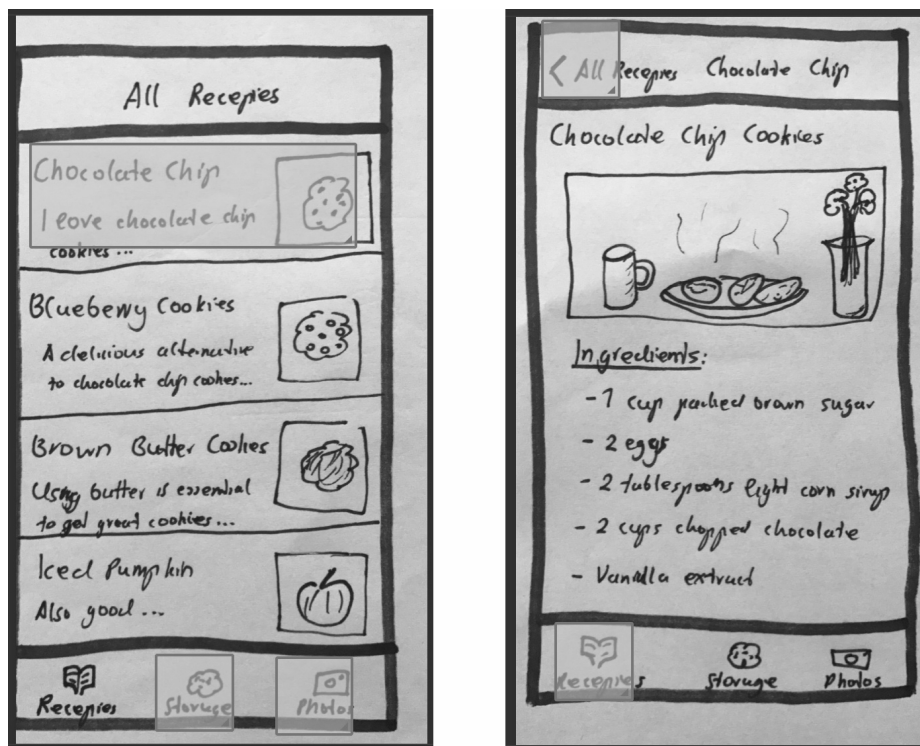


Figure 2.3: Paper prototype digitized using the prototyping tool MarvelApp.

2.3 Continuous Delivery

Continuous Delivery is a software delivery approach introduced by Humble and Farley [HF10] which builds on Continuous Integration [Boo91], [FF06]. With Continuous Delivery, developers not only automatically test each change to a software, but also automate the deployment of each successful tested change to a *target environment* using an repeatable process [HF10]. The characteristics of the target environment, e.g. the operating system

³<https://developer.android.com/studio/>

⁴<https://developer.apple.com/xcode/>

or available software, should match the characteristics of the production environment as closely as possible. The goal is keeping software in a state from which it can be released to the production environment at any point in time. Continuous Delivery workflows are adopted in industry [Kle+15], [Nar15] and are part of software engineering education [KA14] [KABW14], [BKA15].

Humble and Farley’s Continuous Delivery approach is based on tool support for maintaining a *deployment pipeline*. This pipeline, visualized in Figure 2.4, is instantiated every time a developer finishes a development task and commits the changes to the *Version Control Repository*. After each commit the *Commit Stage* (1) is executed, i.e., the change is built and unit-tested. If the commit stage passes successfully, the resulting artifacts are stored in the *Artifact Repository*, from where they can be passed to subsequent *Test Stages* (2). Artifacts are deployed to one or many *Test Environments* (3) which should match the configuration of the *Production Environment*.

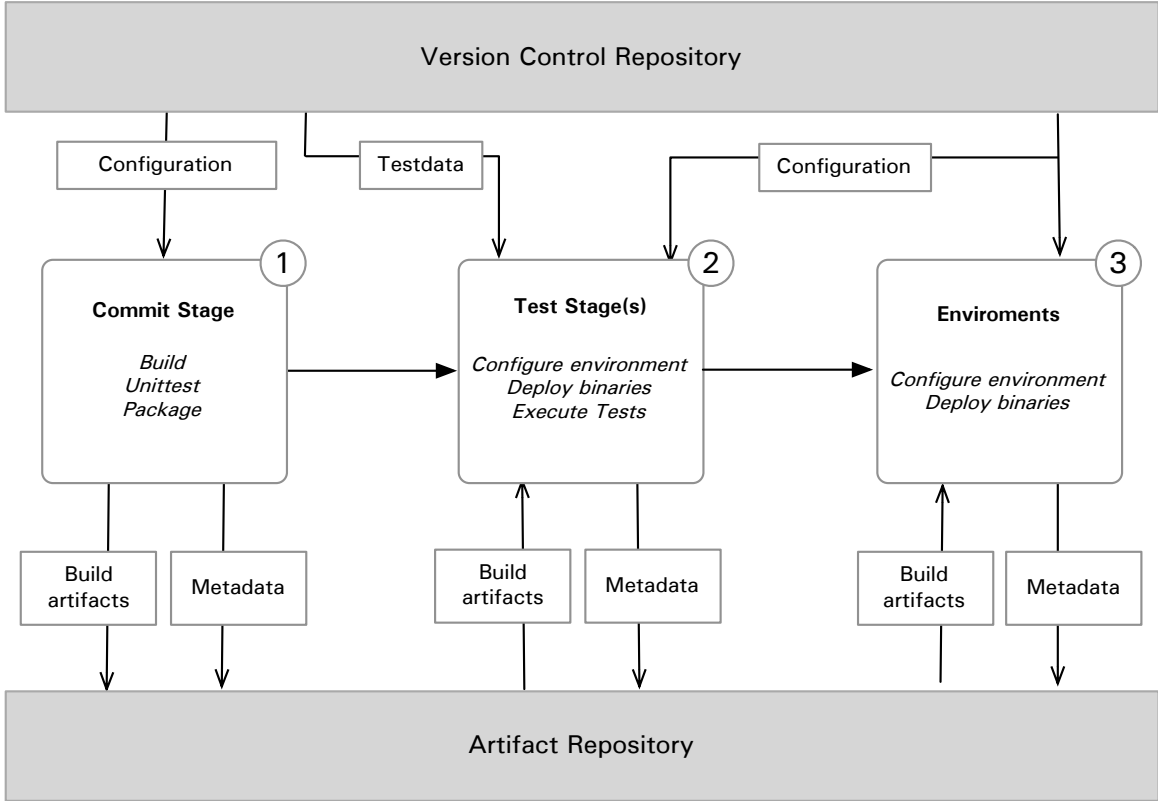


Figure 2.4: Example of a Continuous Delivery pipeline (adapted from [HF10]).

Olseen and Bosch state that by applying Continuous Delivery, software teams are able to "[r]espond and act based on instant customer feedback and see the actual deployment of software functionality as a way of experimenting and testing what the customer needs" [OAB12].

2.4 Continuous Prototyping

In Section 2.2 found that while prototypes are created with mature tool support, none of the tools provides an automated and repeatable process to deliver prototypes the target environment. In Section 2.3 we described Continuous Delivery as a software delivery technique and concluded that while software can be delivered using an automated process, the delivery of prototypes to the target environment has not been dealt with yet.

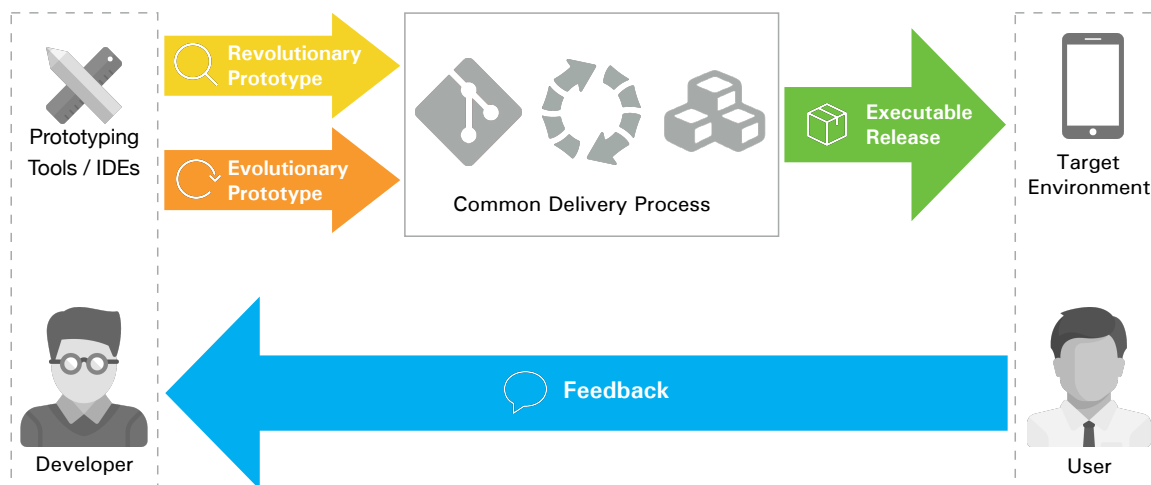


Figure 2.5: Continuous Prototyping approach.

To address this topic, this dissertation coins the term *Continuous Prototyping*. Continuous Prototyping enables developers to deliver revolutionary as well as evolutionary prototypes as executable releases to the target environment at any stage of the development. Stakeholders can evaluate the releases in the target environment and provide feedback to the developers. In order to achieve this, Continuous Prototyping brings the concept of an automated delivery process for software as defined by Humble [HF10] to the field of software prototyping. Figure 2.5 depicts a high-level view of our Continuous Prototyping approach. Continuous Prototyping distinguishes between three kinds of prototypes: *revolutionary prototypes*, *evolutionary prototypes* and *hybrid prototypes*. Figure 2.6 presents a mapping of these prototypes to the techniques described in Section 2.2.1.

Revolutionary prototypes are used to explore and refine requirements and can be created with any technique. After the requirement is discussed such a revolutionary prototype is mostly thrown away. An example for a revolutionary prototype could e.g. be a wireframe prototype created using a prototyping tool.

Evolutionary prototypes are prototypes usually created in the programming language of the proposed system based on already defined requirements. Artifacts of an evolutionary prototype, e.g. source code, are reused and altered in subsequent iterations.

A *hybrid prototype* is an evolutionary prototype which partly consists of revolutionary prototypes. For instance, a team who wants to show off a new user interface design for a part of their application could create a hybrid prototype consisting of the actual application and a mocked user interface for the corresponding part. When development teams adopt Continuous Prototyping, they create revolutionary, evolutionary and hybrid prototypes using a technique of their choice and deliver them using one, common process. Furthermore, they can involve users by delivering executable prototypes to collect feedback already during requirements elicitation.

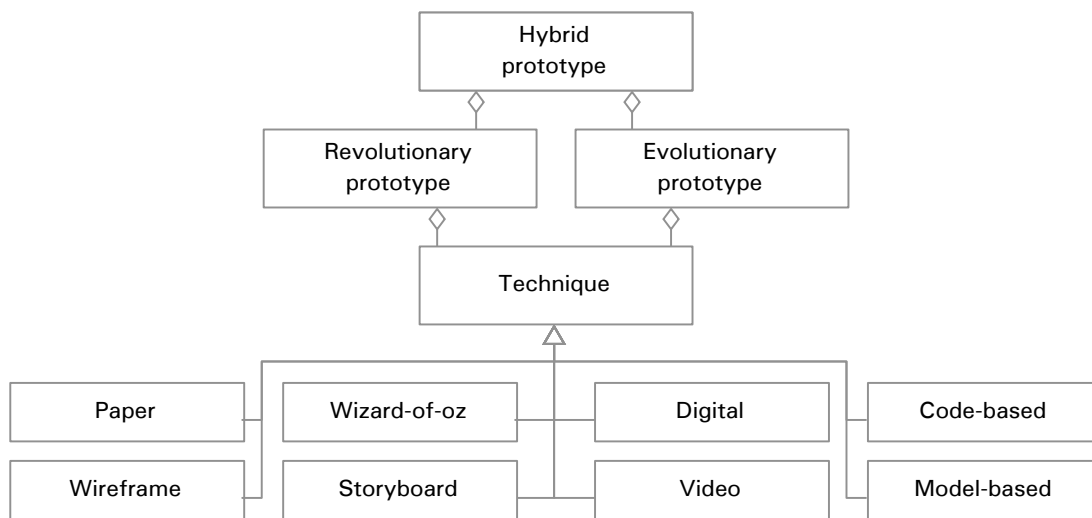


Figure 2.6: Prototypes in Continuous Prototyping - taxonomy (UML Class Diagram).

This chapter presents the ProCeeD Framework. ProCeeD is based on the Continuous Prototyping approach and allows development teams to explore, define and refine requirements by delivering revolutionary, evolutionary as well as hybrid prototypes to the target environment following an automated and repeatable process. ProCeeD enables users to execute and evaluate these prototypes in the target environment. Moreover, ProCeeD allows the concurrent delivery of alternative prototypes to explore multiple design alternatives. Using ProCeeD, developers can clarify requirements by testing hypotheses through collecting implicit and explicit feedback from users. In addition, ProCeeD has the capability to continuously evaluate a team's delivery process based on a set of process metrics.

The remainder of this chapter is structured as follows: Section 3.1 presents six scenarios which describe the interaction of developers and users with ProCeeD. The scenarios are based on three exemplary release plans. Section 3.2 describes the use case model of ProCeeD and its three workflows requirements exploration, delivery automation and process metrics. Based on the scenarios and use-cases Section 3.3 elicits ProCeeD's functional and nonfunctional requirements. Section 3.4 presents ProCeeD's object model and its main objects in detail. Section 3.5 explains ProCeeD dynamic model, in particular its three workflows. Section 3.6 describes ProCeeD components and subsystems. Section 3.7 presents an exemplary mapping of ProCeeD to hardware and software components.

3.1 Scenarios

This section presents ProCeeD's design scenarios using the actors John, a software developer and Carl, a client requesting the application developed by John.

The six scenarios are based on the following, exemplary project setting: John is working on a new feature for an existing interactive system with a particular focus on user interaction design. John creates revolutionary prototypes before implementing a feature to refine and discuss the requirement with Carl. He uses the ProCeeD Framework to manage the creation and delivery of revolutionary and evolutionary prototypes. John regularly delivers releases to Carl, who executes the releases in the target environment and provides feedback to John. Scenario S1 (*Deliver Prototype*) describes how John delivers a prototype to Carl, scenario S2 (*Receive Prototype*) explains how Carl is notified after a prototype has been released, how Carl can access a release and how he provides feedback. In scenario S3 (*Deliver Multiple Prototypes*) John leverages ProCeeD to show Carl multiple design

alternatives, scenario S4 (*Deliver Hybrid Prototypes*) presents how ProCeeD allows John to combine revolutionary and evolutionary prototypes into a hybrid prototype. Scenario S5 (*Decide On Release Schedule*) describes how John uses ProCeeD to define process metrics. Finally, scenario S6 (*Analyze the Release Process*) describes how John uses metrics to monitor the release process of his team.

3.1.1 Deliver Prototype

Scenario S1 and S2 are based on the exemplary release plan depicted in Figure 3.3. The release plan is structured as follows: The *Development* section depicts each iteration John created for a new feature. Iteration RV1-RV5 are revolutionary prototypes, iteration EV6-EV8 are carried out using evolutionary prototypes and are therefore implemented in the programming language of the target system. The section *Groups and Releases* show the three target groups developers, clients and users. Carl can deliver releases to any of the three groups. In addition it shows the releases ("Rx") created by John and delivered to the groups during the development of the new feature. Yellow Releases are based on revolutionary prototypes, orange releases are based on evolutionary prototypes. Feedback by a member of a group is depicted by a blue circle.

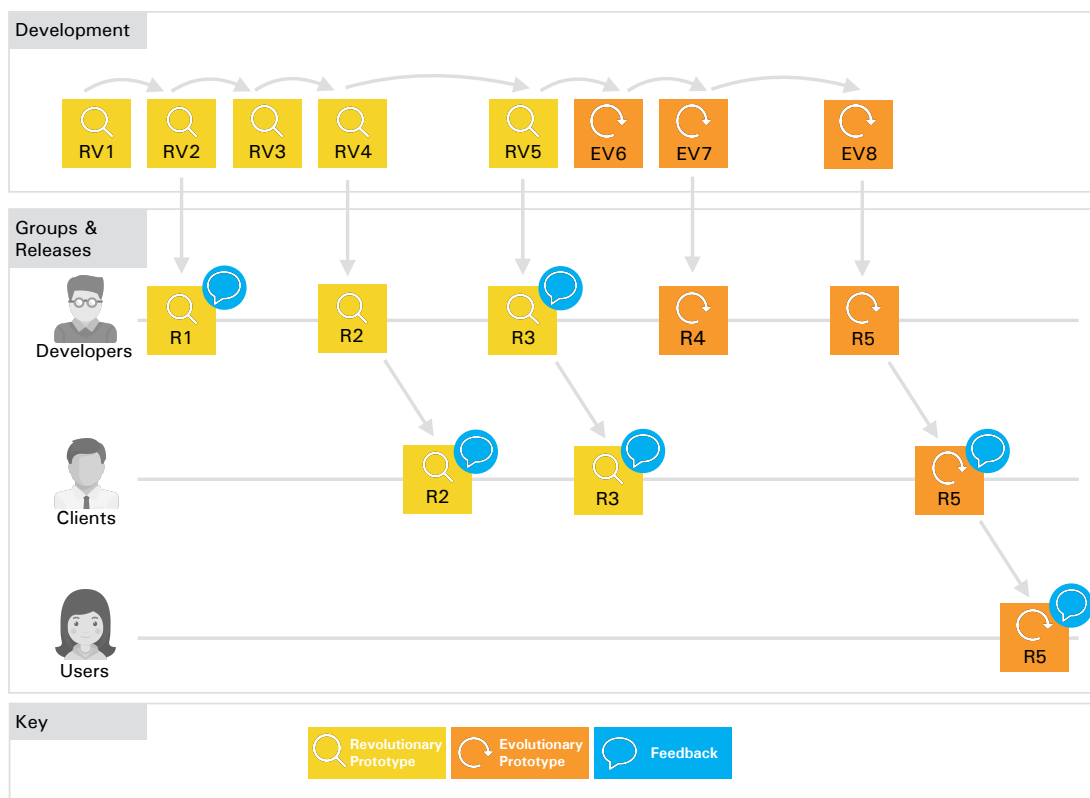


Figure 3.1: Release plan for scenario S1 and S2.

The following scenario describes how a developer *John* uses ProCeeD to deliver executable prototypes to a client *Carl*.

Scenario name DeliverPrototype

Participating actor instances john:Developer, carl:Client

Flow of events

1. *John* finishes an iteration of a revolutionary prototype for a requirement "User Login using Password Manager" using the web-based prototyping tool MarvelApp. He now wants to deliver the prototype to the other members of the development team to obtain feedback.
2. *John* uses ProCeeD to transform and package the revolutionary prototype into an application which can be executed in the target environment of the project, a mobile application for the iOS platform.
3. He creates a release ("R1") from the revolutionary prototype to deliver it to a user group.
4. He writes short release notes to describe how the new user interface allows the integration of a password manager.
5. *John* delivers the release to all members of the development team and asks for their feedback.
6. After reviewing the feedback on the new login functionality from his team, *John* implements a couple of refinements and delivers a next release ("R2") to his team.
7. The team agrees to ask *Carl* for his feedback. *John* therefore promotes R2 to *Carl*.
8. ProCeeD notifies *Carl*, who can now install the release on his phone. He provides feedback using a button inside the application.
9. After another release ("R3"), *John* implements the interaction design from the revolutionary prototype into the evolutionary prototype ("EV6" to "EV8").
10. After *John* accepted the implementation in R5, the release is promoted to a group of end users for further evaluation.

3.1.2 Receive Prototype

The following scenario describes how Carl receives a release delivered by John.

Scenario name ReceivePrototype

Participating actor carl:Client
instances

Flow of events

1. *Carl* is on the way home from work when he receives an email from *John*, one of the developers of a project he is involved in as a client.
 2. *Carl* reads the email notification which states that a new releases has been delivered to him.
 3. He reviews the release notes *John* wrote about the integration of a password manager to improve the login experience.
 4. *Carl* accesses the new release on his mobile device using a link in the email.
 5. He executes the release on his mobile phone, focusing on the login feature *John* highlighted in the release notes.
 6. *Carl* taps the feedback button inside the app and provides feedback to *John* about a design issue with the login button and a logical inconsistency he found in the interaction model when he forgot to enter a password.
 7. The next morning *Carl* shows the prototype to some colleagues and asks for their feedback.
 8. He adds the feedback by his colleagues via the feedback button and sends it to *John*.
-

3.1.3 Deliver Multiple Prototypes

ProCeed should allow developers to deliver of multiple prototypes to a group of users at the same time. Scenario S3 describes this functionality and is based on the release plan depicted in Figure 3.3.

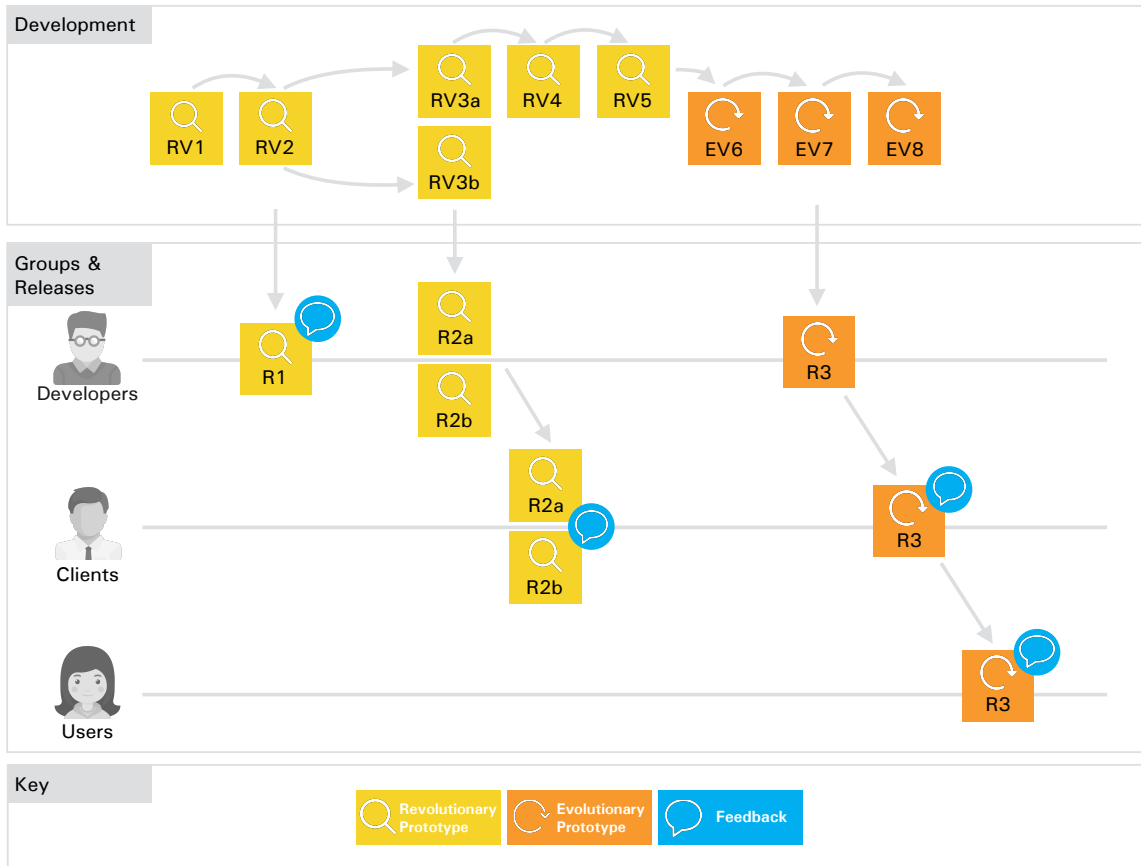


Figure 3.2: Release plan for scenario S3.

The release plan is structured as follows: The *Development* section presents each iteration the development team went through for a new feature. Iteration RV1-RV5 are revolutionary prototypes, iteration EV6-EV8 are evolutionary prototypes and therefore implemented in the programming language of the target application. For Iteration RV3, two design alternatives, (RV3a and RV3b), are created and delivered. Section *Groups and Releases* shows the releases ("Rx") created and delivered during the development of the new feature. Yellow releases are based on revolutionary prototypes, orange releases are based on evolutionary prototypes. The visualization contains a blue feedback icon for releases that have received feedback from at least one member of the corresponding group.

The following scenario describes how the developer *John* uses ProCeeD to clarify a requirement by delivering multiple prototypes to the client *Carl*.

Scenario name DeliverMultiplePrototypes

Participating actor instances john:Developer, carl:Client

Flow of events

1. *John* is preparing the user interaction design for a requirement regarding the management of the user profile. After he finished a first draft, he creates a release R1 and delivers it to his fellow developers.
 2. He and his team conclude that they could implement two solutions which differ in the amount of interaction steps and complexity. One alternative would only show the contents of the user profile, the other alternative would allow a user to edit their user profile inside the application.
 3. Using the prototyping tool *MarvelApp* *John* creates two revolutionary prototypes, (Ex3a and Ex3b) to demonstrate the possible alternative solutions.
 4. *John* decides that he needs to ask *Carl* for feedback with which alternative to continue. He creates a release for each of the revolutionary prototypes, R2a and R2b.
 5. In addition, he writes release notes for both prototypes. He includes questions he wants *Carl* to answer while he is evaluating and comparing both design alternatives.
 6. He promotes both releases to *Carl* at 6pm.
 7. *Carl* is already on its way home when he receives an email notification about the new releases.
 8. Using the link in the email, he can access both prototypes without remembering his credentials. *Carl* spends his subway trip evaluating the two prototypes.
 9. A day later the team meets with *Carl* who has already evaluated the two releases on his device.
 10. After a short discussion, the team and *Carl* decide for the simpler alternative without the ability to edit the user profile inside the application.
 11. *John* implements the requirement in the evolutionary prototype.
-

3.1.4 Deliver Hybrid Prototype

The following scenario describes how *John* uses ProCeeD to create a hybrid prototype based on an evolutionary prototype and components of a revolutionary prototype. A hybrid prototype contains an actual implementation of some parts of an application, while the other parts are still a revolutionary prototype. This allows developers to quickly evaluate an idea or a possible change to the user interaction design in the context of the complete system. It also allows developers to present each iteration within the context and scope of the application in development. A user can experience both parts at once, the already implemented parts as well as the parts which are still prototypical, e.g. in the form of a wireframe or a digital prototype.

The following scenario S4 is based on the exemplary release plan depicted in Figure 3.3. The release plan is structured as follows: The *Development* section presents each iteration the development team created for a new feature. Iteration RV1-RV2 are revolutionary prototypes. For the iterations HY3-HY5, John creates a hybrid prototype consisting of an evolutionary prototype and parts of the revolutionary prototype. Iteration EV6-EV8 are carried out using evolutionary prototypes.

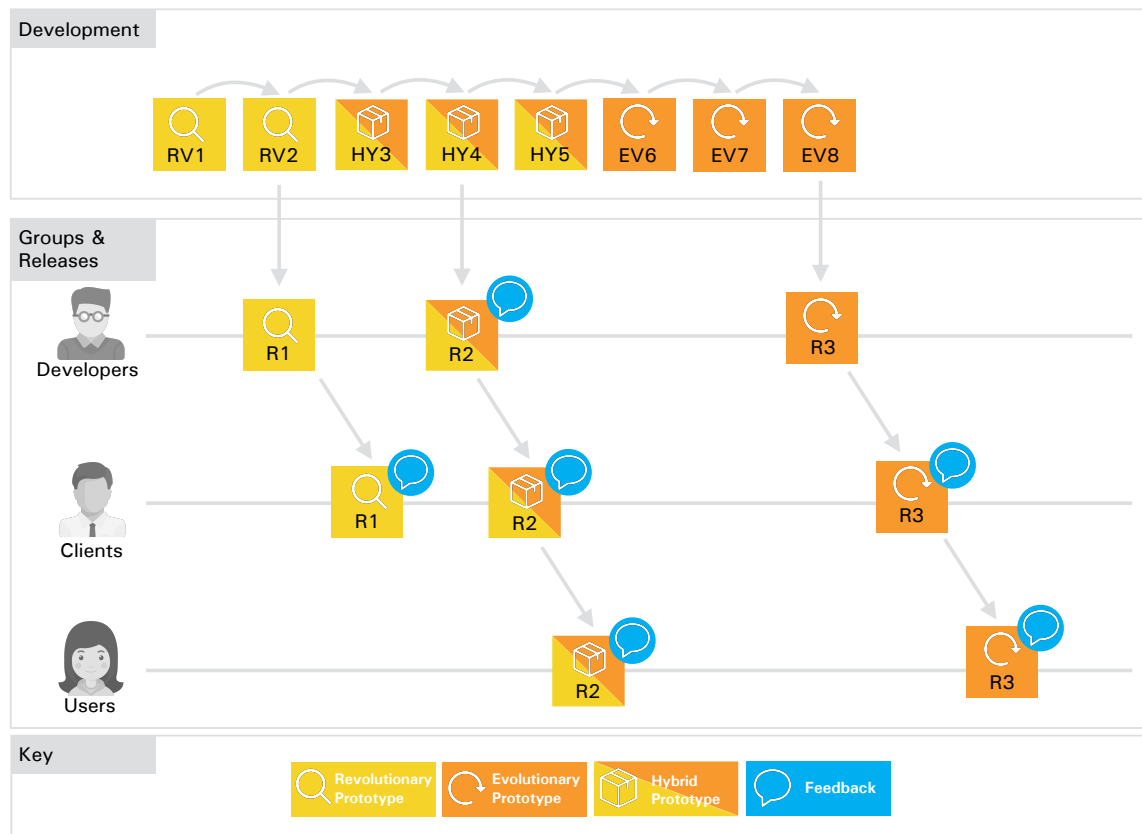


Figure 3.3: Release plan for scenario S4.

Scenario name DeliverHybridPrototype

Participating actor instances john:Developer, carl:Client

Flow of events

1. *John* creates a revolutionary prototype for a change to the user interaction design of the application's onboarding workflow. He creates a release ("R1") and promotes it to *Carl*.
 2. *Carl* sends feedback to *John* that he needs to see the new user interface in the context of the full application in order to evaluate it properly.
 3. *John* uses ProCeeD to replace parts of the evolutionary prototype with the revolutionary prototype he just created.
 4. *John* creates a hybrid release R2.
 5. He promotes the release ("R2") to *Carl*.
 6. *Carl* likes the new design and decides to ask some clients for their feedback.
 7. *John* refines the release notes so that the clients understand what has changed in the release and promotes the R2 to the client group.
 8. After receiving positive overall feedback, *John* implements the requirement.
 9. *John* finishes the implementation and delivers a release R3 to *Carl* and later to the users.
-

3.1.5 Decide on a Release Schedule

After presenting scenarios covering the requirements exploration and delivery automation workflows of ProCeeD, we now describe a scenario which involves ProCeeD's process metrics workflow. The following scenario S5 presents how a Developer *John* uses ProCeeD to decide when to send the next release to *Carl*. This allows John to regularly send releases to *Carl* without overwhelming him.

<i>Scenario name</i>	<u>DecideOnReleaseSchedule</u>
----------------------	--------------------------------

<i>Participating actor instances</i>	<u>john:Developer, carl:Client</u>
--------------------------------------	------------------------------------

Flow of events

1. *John* did some refinements to a prototype he is currently working on in collaboration with *Carl*.
 2. *John* delivered a release of an earlier version of the prototype to *Carl* a day ago but hasn't received any feedback yet.
 3. *John* wants to find out if *Carl* has already accessed the release and how long *Carl* usually takes to access a release.
 4. He uses ProCeeD to define a corresponding metric called "Time between release created and release accessed".
 5. Using the metric he finds out that *Carl* typically reviews new releases after 48h.
 6. *John* decides not to send the next release until *Carl* has given feedback for the already delivered one.
 7. After four days and two days later as usual, *Carl* has still not replied, so *John* reminds him that he needs his input on the release.
 8. The next morning he receives an email from *Carl* with detailed feedback.
 9. *John* finds *Carl's* feedback useful and incorporates it before delivering another release to *Carl*.
-

3.1.6 Analyze the Release Process

Scenario S6 shows how ProCeeD helps *John* to get a better understanding of the performance and bottlenecks of his team's release process.

Scenario name AnalyzeReleaseProcess

Participating actor instances john:Developer, carl:Client

Flow of events

1. In a team meeting, *John* discusses why it is important to regularly evaluate releases in the target environment.
 2. *John* explains that the three new members who joined the team in the last weeks are not yet used to the practice. The team agrees on being extra careful in sticking to the workflow.
 3. To measure their progress, *John* uses ProCeeD to define a *release frequency* metric and adds it to the team dashboard.
 4. He introduces the metric in the next development team meeting.
 5. From now on the team will review the metric in every team meeting and discuss possible ways to improve.
 6. After four weeks, *John* has a look at the metrics with the whole team. The metrics dashboard shows that the team has delivered a release to the target environment at least once per week.
 7. *John* discusses the positive result with the team. Together they agree to stick with the release schedule.
-

3.2 Use Cases

Based on the scenarios described in Section 3.1 we present ProCeeD's use case model for the three workflows delivery automation, requirements exploration and process metrics. Each model depicts the use cases and their relationships for the actors *Developer* and *User*. A *User* can be any person who will use the proposed system, e.g. the client or an end user, but also a developer.

Requirements Exploration

Figure 3.4 shows the use case model for the requirements exploration workflow. A *Developer* can deliver different release alternatives to a group of *Users*. He can also manage the feedback of the users by storing and analyzing it in an issue tracking system.

A *User* can access a release, evaluate it in the target environment, and provide feedback. He can compare different releases while executing them in the target environment, e.g. to decide on aspects of the interaction design. While using a release, the *User* can give in-situ feedback at any time. He can give textual feedback or annotate a screenshot before sending it to the *Developers*.

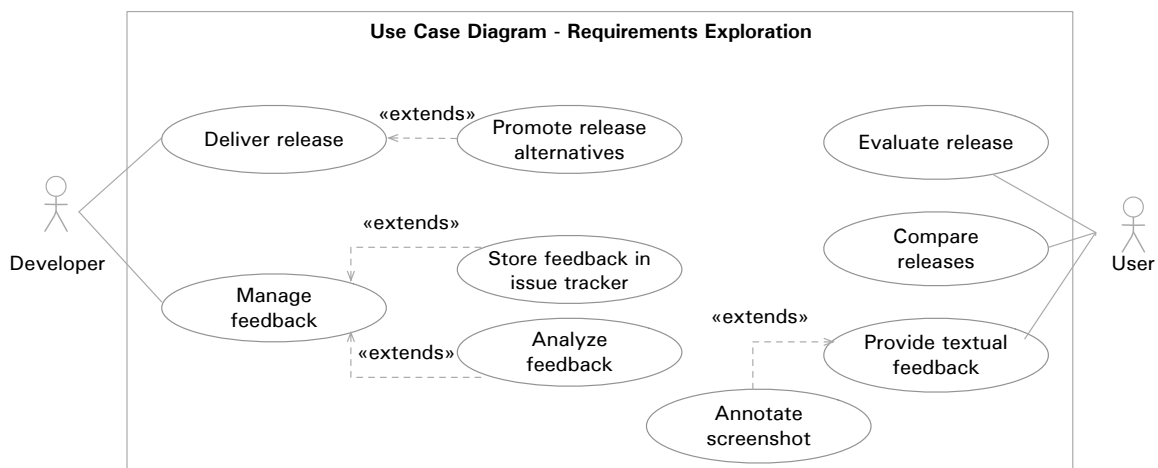


Figure 3.4: ProCeeD's requirements exploration use case model (UML Use Case Diagram).

Delivery Automation

Figure 3.4 shows the use case model for the delivery automation workflow. A *Developer* can create a new release from revolutionary as well as evolutionary prototypes. He can provide textual release notes for a release and deliver it to e.g. a group of *Developer*. Afterwards, he can promote the release, which is currently delivered to the development team, to a group of clients. Finally he can determine the target audience of a release by managing the groups of *Users* who can access a respective release.

A *User* can access a release which has been released to a group he is a member of. He can read the release notes for the release as provided by the *Developer*.

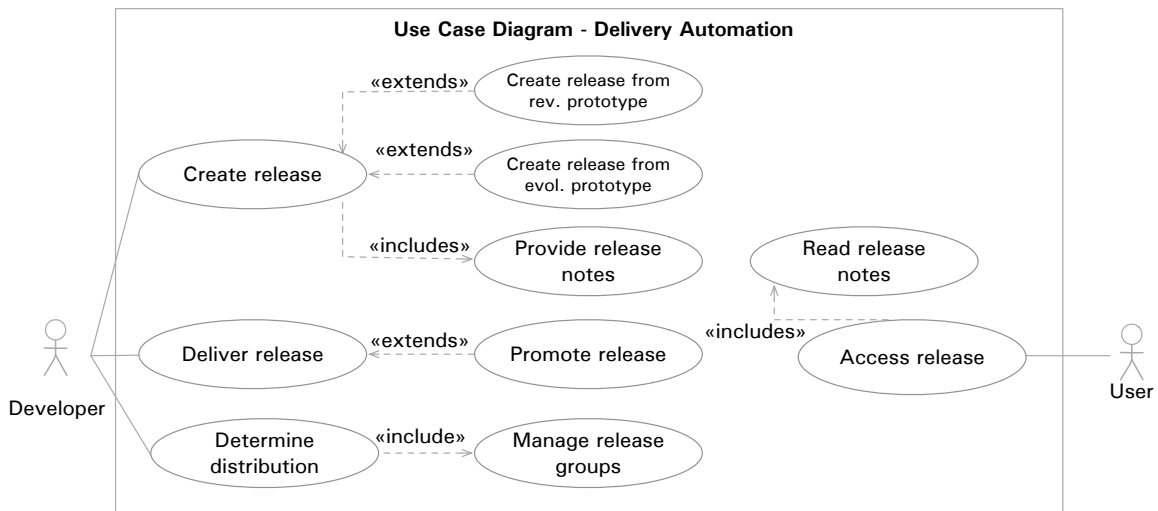


Figure 3.5: ProCeeD’s delivery automation use case model (UML Use Case Diagram).

Process Metrics

Figure 3.4 shows the use case model for the process metrics workflow. A *Developer* invokes an event every time he creates or delivers a release. A *User* invokes an event every time he accesses a release or when he sends feedback to the developer.

Based on these events, a *Developer* can define process metrics which represent time spans between the events. Developers can use the metrics to derive decisions about e.g. when to send the next release to a user group or when to ask a client for additional feedback.

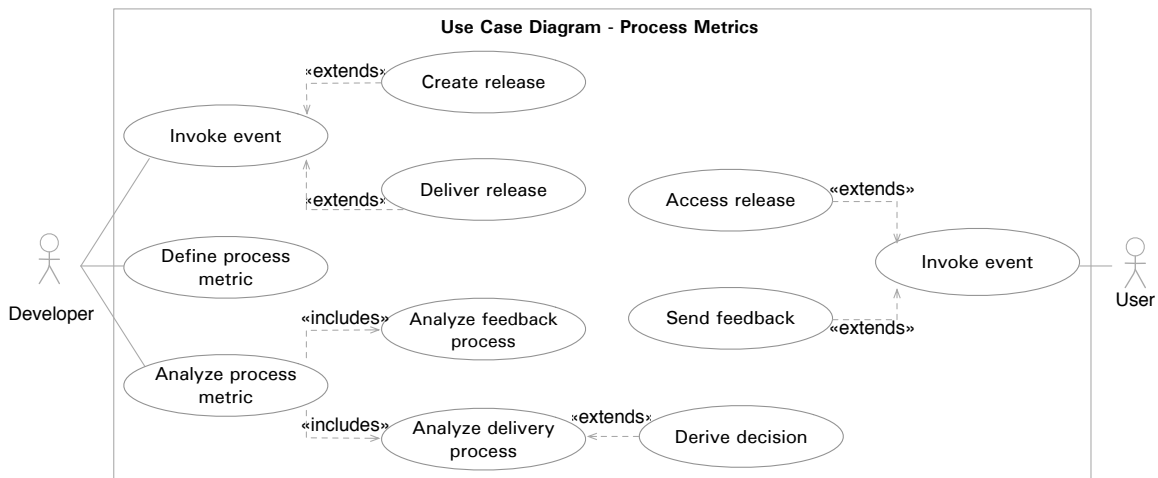


Figure 3.6: ProCeeD’s process metrics use case model (UML Use Case Diagram).

3.3 Requirements

Based on the scenarios and use cases, we now derive ProCeeD's functional and nonfunctional requirements [BD09]. The functional requirements are grouped depending on which of the three workflows delivery automation, requirement exploration and process metrics they are part of.

The requirements exploration workflow enables developers to explore requirements by creating one or more prototypes of the proposed system. The delivery automation workflow allows developers to deliver revolutionary and evolutionary prototypes based on the Continuous Prototyping approach described in Section 2.4. Users can execute and evaluate the prototypes in the target environment and give feedback to define or refine requirements. The process metrics workflow allows the measurement of process metrics to improve the adoption of the ProCeeD Framework inside development teams. In addition, the workflow allows developers to decide on the release schedule based on the collected data.

3.3.1 Functional Requirements

In this section we describe ProCeeD's functional requirements using the two actors *Developer* and *User*.

Requirements Exploration

- FR1.1** ProCeeD shall be able to deliver multiple releases to a group of users at the same time, e.g. in order to get a decision or feedback on several design alternatives.
- FR1.2** ProCeeD shall allow a user to give textual feedback to the developer while using the application in the target environment. For instance, if the user evaluates a mobile application prototype, he should be able to state feedback on any screen of the application.
- FR1.3** ProCeeD shall allow a user to create and annotate screenshots of the user interface while executing and evaluating a release and attach them to a feedback item.
- FR1.4** ProCeeD shall allow the management of explicit and implicit feedback provided by users. For each feedback item, the author and timestamp of the feedback should be collected. Feedback should be managed on a release level.

Delivery Automation

- FR2.1** ProCeeD shall be able to deliver a release of a revolutionary prototype to the target environment of the system in development.
- FR2.2** ProCeeD shall accommodate the necessary transformation steps to allow a revolutionary prototype to be executed in the target environment.
- FR2.3** ProCeeD shall be able to deliver a release of an evolutionary prototype to the target environment.
- FR2.4** ProCeeD shall allow the promotion of a release to a group of users.
- FR2.5** ProCeeD shall notify each user with the appropriate permissions about a new release. The notification should contain the version of the release, the release notes as stated by the developer and a link to access the release.
- FR2.6** ProCeeD shall allow developers to manage users and assign them to user groups. Managing users includes the invitation of new users and the removal of existing users from user groups.
- FR2.7** ProCeeD shall allow developers to provide release notes for a release. Developers shall be able to refine release notes during each promotion step.
- FR2.8** ProCeeD shall allow developers to look up which user group has or had access to which release(s) at any point in time.

Process Metrics

- FR3.1** ProCeeD shall gather the following time stamps for each release: when did a developer create a release; when did a user receive a release; when did a user access a release; when did the user provide feedback on a release.
- FR3.2** ProCeeD shall allow a developer to compute process metrics and provide the capability to analyze them on a user group level. ProCeeD should visualize the process metrics in a way that a developer can analyze the duration of each step of the delivery process.
- FR3.3** ProCeeD shall allow a developer to look up when a user received which release(s) and when the user gave feedback.
- FR3.4** ProCeeD shall allow a developer to check what prototype a certain release contains.
- FR3.5** ProCeeD shall allow a developer to look up which user accessed which release(s) at any given point in time.

3.3.2 Nonfunctional Requirements

For ProCeeD we derive the following nonfunctional requirements, categorized by usability, interface, supportability and performance [BD09].

Usability

NFR1.1 ProCeeD shall allow developers to apply the same delivery workflow and interaction steps for the delivery of revolutionary as well as evolutionary prototypes.

NFR1.2 ProCeeD shall support a fast workflow for developers to create and deliver a new release.

NFR1.3 ProCeeD shall support a fast workflow for users to provide explicit feedback.

NFR1.4 ProCeeD shall allow a developer to monitor the release process of multiple applications at once quickly.

Interface

NFR2.1 ProCeeD shall be able to retrieve revolutionary prototypes from prototyping tools. At least two prototyping tools need to be supported.

NFR2.2 ProCeeD shall retrieve artifacts like source code of evolutionary prototypes from a version control system.

NFR2.3 ProCeeD shall allow developers to store feedback in an issue tracking system. Only one issue tracking system needs to be supported at the same time.

NFR2.4 ProCeeD shall be able to control and monitor the build process of a continuous integration service.

Supportability

NFR3.1 ProCeeD shall support multiple target environments. For example if a mobile application is developed for the Android and iOS ecosystems, developers should be able to deliver revolutionary as well as evolutionary prototypes to both environments.

NFR3.2 ProCeeD shall allow the integration of a user feedback service.

NFR3.3 Process metrics collected by ProCeeD shall be visualized on a user group level.

Performance

NFR4.1 ProCeeD should allow developers to access the system even while a high amount of users access releases.

NFR4.2 ProCeeD should be responsive even when connected systems are inaccessible. These are: the integration service, prototyping tools and the issue tracker.

3.4 Object Model

In this section we present ProCeeD's object model. Figure 3.7 depicts the objects of the problem domain and their methods and attributes.

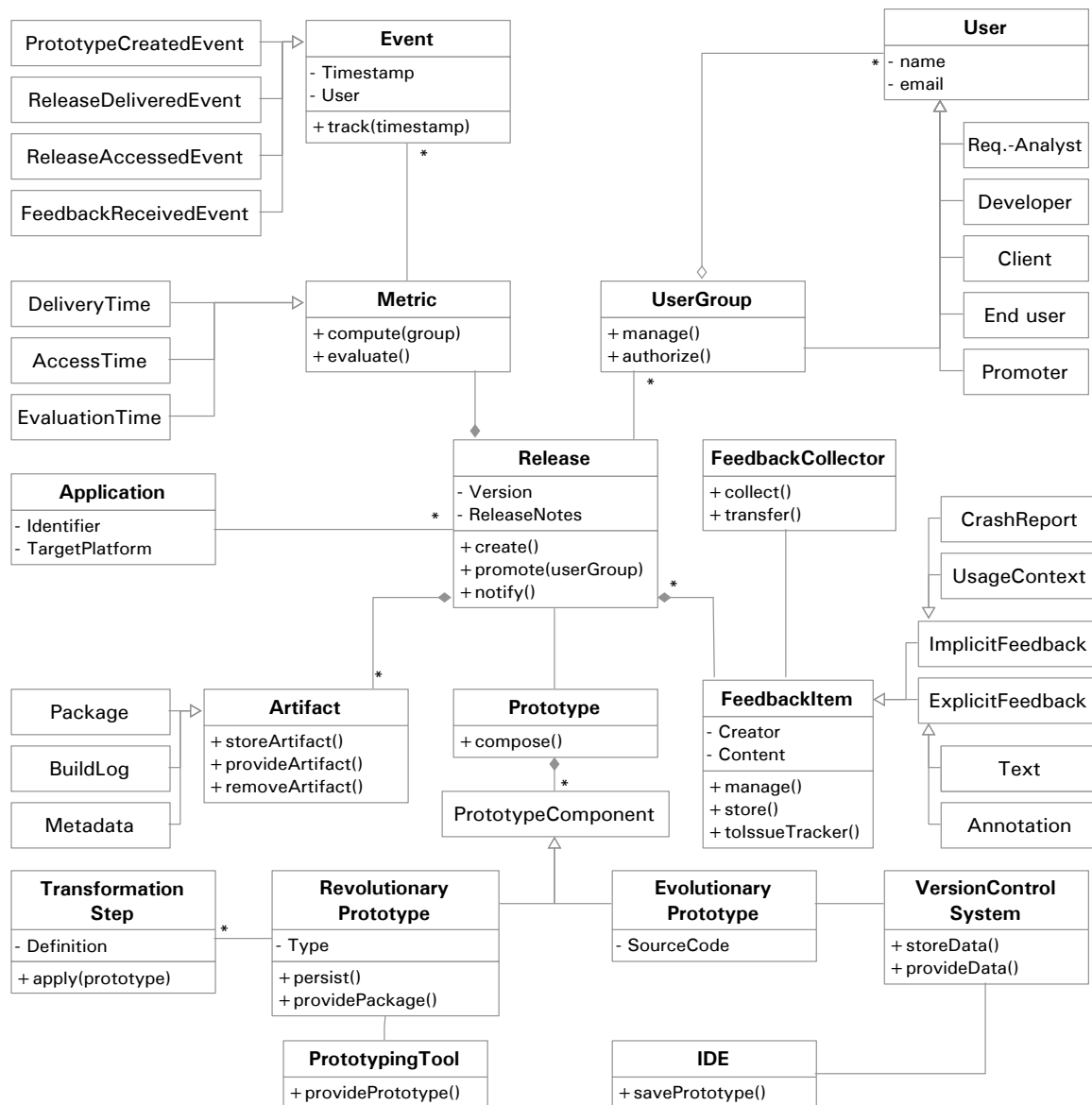


Figure 3.7: ProCeeD's object model (UML Class Diagram).

The model is structured as follows: an Application has 0..n Releases. A Release is based on one Prototype which is composed of one or more PrototypeComponents. PrototypeComponents are either a RevolutionaryPrototype or an EvolutionaryPrototype.

A RevolutionaryPrototype is created using a PrototypingTool and by applying a technique described in Section 2.2. For each RevolutionaryPrototype 0..n TransformationSteps are modeled. ProCeeD applies the TransformationSteps on the RevolutionaryPrototype to allow its delivery as an executable Release. A RevolutionaryPrototype is usually created to collect feedback on the interaction model or user interface design of a requirement.

An EvolutionaryPrototype is created based on artifacts like source code developed using the IDE and programming language of the target system. Developers add and alter source code in the VersionControlSystem when implementing a requirement. They create a Release every time they need to collect feedback or when the implementation of a requirement is finished.

A developer can choose to compose a prototype of an EvolutionaryPrototype and parts of a RevolutionaryPrototype. Such a prototype, consisting of both kinds of PrototypeComponent, is called hybrid prototype. A hybrid prototype is delivered if e.g. parts of the user interface need to be created or reworked or a new feature is shown best in the context of the complete application. The hybrid prototype is modeled as a Prototype object in our model.

Each Release created from a Prototype is identified by a Version. A Release is composed of 1..n Artifacts. These are e.g. the executable AppPackage created from the compiled source code or BuildLogs which document the result of the build process of a Release. Metadata, describing e.g. a timestamp when a certain Release was created, is also stored as an Artifact. A developer can provide ReleaseNotes for a Release in textual form.

A Release can be promoted to a UserGroup consisting of 0..n Users. Users are the persons executing a release, e.g. a Requirements Analyst, a Developer, a Client, a End user or a Promoter. ProCeeD manages which UserGroup has access to which Release. After a Release got promoted to a UserGroup, all Users of the UserGroup can access it.

Users can provide FeedbackItems for a Release. A FeedbackItem is described by its creator and content. ProCeeD allows developers to collect and manage ExplicitFeedback like Textual feedback or Annotations by a User. In addition, ProCeeD allows developers to collect and manage ImplicitFeedback like Usage Context and Crash Reports for each release.

For each release, ProCeeD tracks Events which are triggered by action carried out by members of a UserGroup. Supported Events are PrototypeCreated, ReleaseDelivered, ReleaseAccessed, FeedbackReceived. Based on the Events ProCeeD can compute process Metrics for each Release. Metrics supported by ProCeeD are DeliveryTime, AccessTime and FeedbackTime.

3.5 Dynamic Model

This section presents the dynamic model of ProCeeD. We present a state machine diagram of a ProCeeD Release object and provide a model as well as a detailed description of the ProCeeD workflows requirements exploration, delivery automation and process metrics.

3.5.1 Release

Figure 3.8 shows the dynamic model of a ProCeeD release in a UML State Machine Diagram depicting all states a release can adopt as well as the transitions between the states.

After a developer created a release from a prototype, its state transitions to created. When a developer decides to release to a group of users, it transitions into the delivered state. As soon as a member of a user group accesses the release in the target environment for the first time, the state changes to received. When a user provides feedback, the release transitions to commented. A release can be replaced when a developer delivers the next release to the user group. If a release is either in the delivered, received or commented state and is replaced by a subsequent release, it transitions to its final state, replaced.

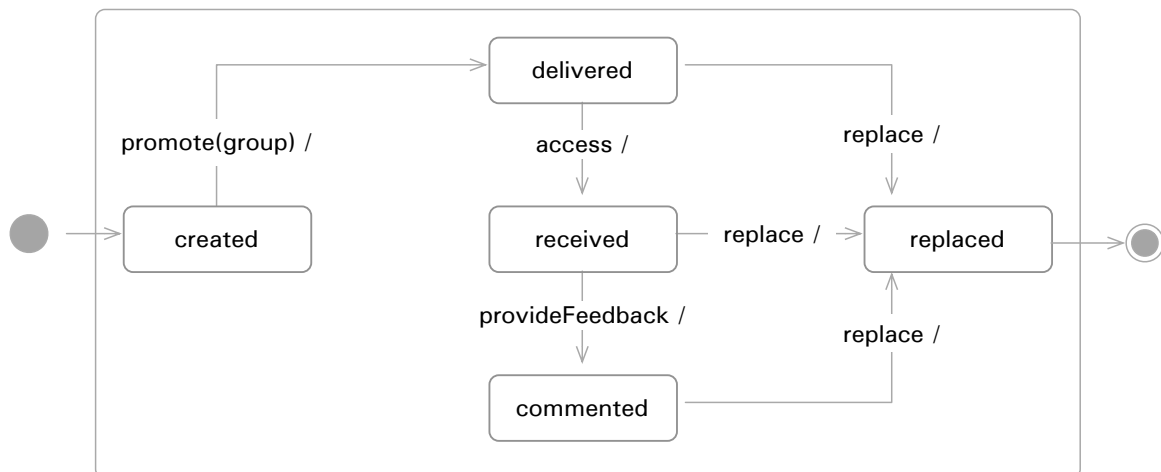


Figure 3.8: ProCeeD's release model (UML State Machine Diagram).

This release state machine is the foundation for the three ProCeeD workflows described in the next section. In particular, the different states of a release as well as their transitions are recorded as events and later used for the computation of metrics in the process metrics workflow.

3.5.2 Workflows

This section describes ProCeeD's workflows requirements exploration, delivery automation and process metrics. An overview is depicted in Figure 3.9.

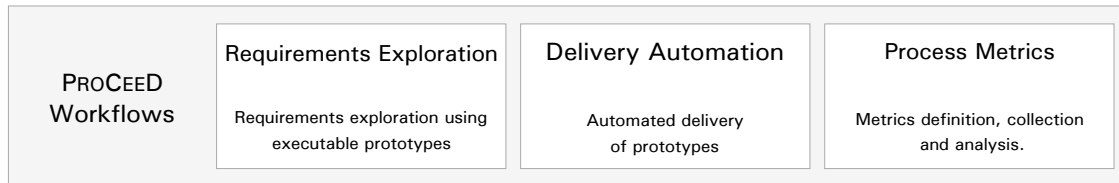


Figure 3.9: ProCeeD's workflows.

The requirements exploration workflow enables developers to elicit and explore requirements by creating one or multiple prototypes of the proposed system. The delivery automation workflow allows developers to deliver revolutionary as well as evolutionary prototypes. Users can execute and evaluate the prototype in the target environment and give feedback. The process metrics workflow allows developers to measure and evaluate their teams delivery workflow.

Requirements Exploration

Using ProCeeD's requirements exploration workflow developers can apply an iterative and incremental prototyping approach. Figure 3.10 shows the workflow as a UML Activity Diagram.

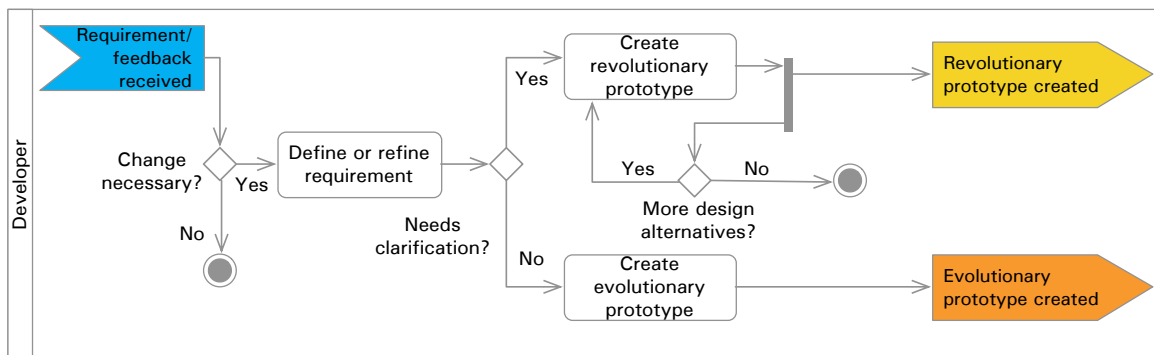


Figure 3.10: ProCeeD's requirements exploration workflow (UML Activity Diagram).

The workflow is executed every time a developer receives a new requirement or feedback on a existing requirement. A developer first decides whether the feedback leads to a new or refined requirement. If the feedback does not impact the requirements of the project, the workflow terminates. Depending on the maturity of the requirement, the developer now decides if it can be implemented as a change to the evolutionary prototype or if a revolutionary prototype needs to be created to collect more feedback and clarify the requirement before its implementation. An evolutionary prototype is usually developed using the IDE and programming language of the target environment, while a revolutionary prototype can also be created using a prototyping tool. The workflow repeats and the

revolutionary prototype is refined until the requirement is implemented or dismissed. If design alternatives for a requirement exist, ProCeeD allows developers to create multiple prototypes for each alternative of the requirement. Users can then evaluate and compare the prototypes and decide which alternative to implement. For instance, the workflow can be used to decide between different user interface designs or interaction models for a requirement. The decision process of which alternative to choose is beyond the scope of this dissertation.

Delivery Automation

Using the delivery automation workflow, a developer can automate the deployment and delivery of both revolutionary and evolutionary prototypes. Figure 3.11 shows an overview of this workflow, which is executed every time a prototype was created and a developer wants to deliver the prototype in order to collect feedback.

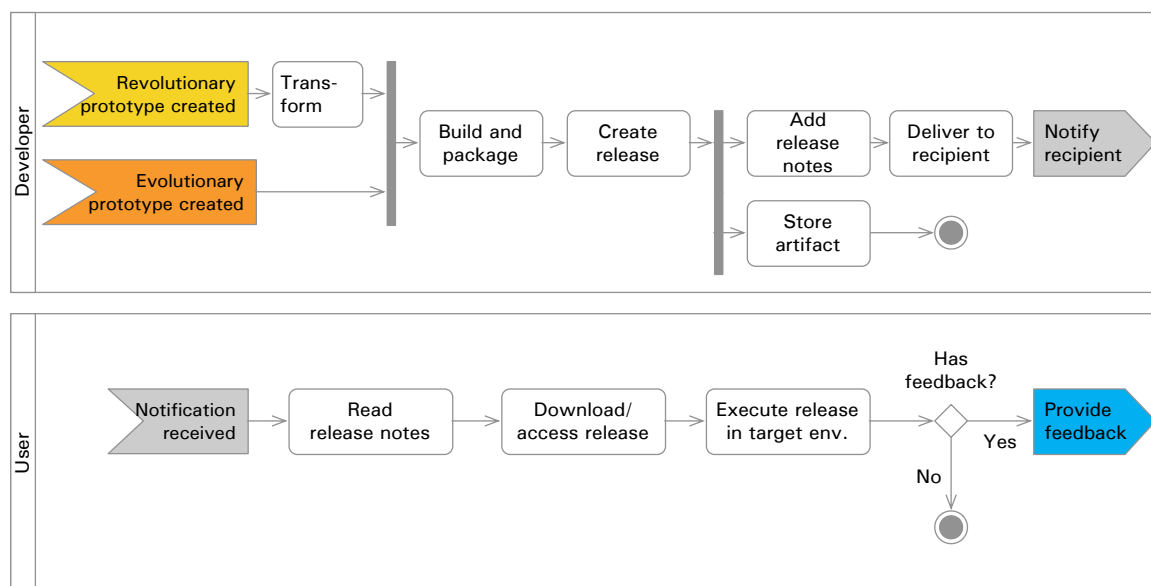


Figure 3.11: ProCeeD's delivery automation workflow (UML Activity Diagram).

In a first step, ProCeeD transforms revolutionary prototypes to be executable in the target environment using predefined transformation steps for the respective prototyping tool used by the developer. Following this, the prototype is built and packaged. The resulting artifacts are stored in the artifact repository of the project. A developer can now create a release and define release notes. A release can be delivered to a user group, and as soon as this happens, the members of the group are notified. Each user in the user group can now access the release and execute it in the target environment. ProCeeD allows developers to include user feedback and usage analytics components into releases created from revolutionary and evolutionary prototypes. These allows a user to give in-situ feedback while using the release.

Process Metrics

With ProCeed's process metrics workflow, developers can analyze and adjust their team's delivery process. Figure 3.12 shows the workflow as a UML Activity Diagram.

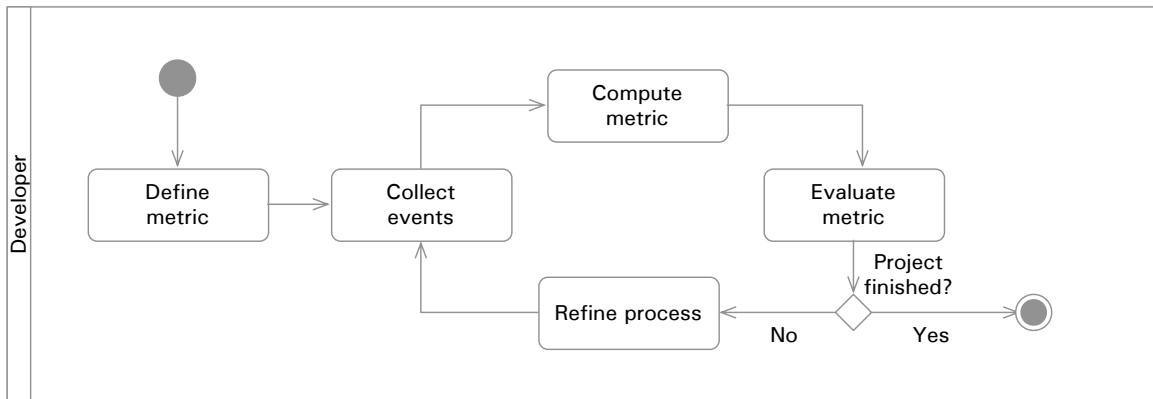


Figure 3.12: ProCeed's process metrics workflow (UML Activity Diagram).

ProCeed collects the following events during the lifecycle of a prototype: PrototypeCreated, ReleaseDelivered, ReleaseAccessed and FeedbackReceived. The events are described in the object model shown in Section 3.4.

A developer can define a process metric based on the events. ProCeed collect appropriate events and computes the metric e.g. for each release. A developer can evaluate a metrics and derive e.g. refinements for the release process. Figure 3.13 gives an overview of the metrics ProCeed provides to developers.

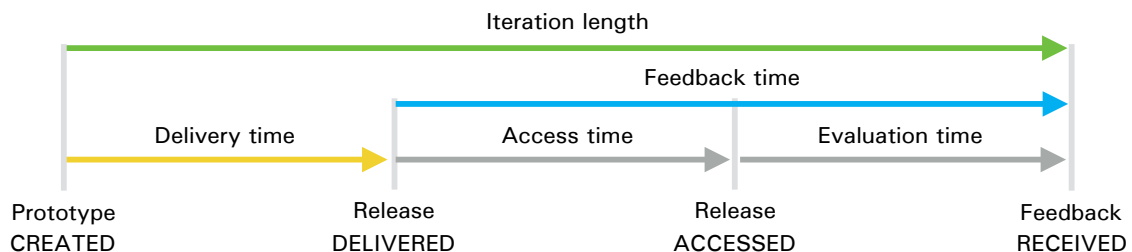


Figure 3.13: ProCeed's process metrics - example

We define a Prototyping Iteration as the time span between the creation of a prototype and receiving feedback on the prototype. An iteration can be divided into three parts: The Delivery Time is defined as the time span between the creation of a prototype and its delivery to a user group. The Access Time reflects the time a user takes to access a new release after it was delivered. The Evaluation Time measures the time span between the moment a user accessed a release for the first time and the moment the user provides feedback to the development team.

Developers can use ProCeeD's process metrics to evaluate their delivery workflow. For example, a team that develops a new feature could define the following metric: using ProCeeD they can calculate the time between a release was delivered to their client and the time the client accessed the release for the first time in the target environment. Using the metric, the team can now decide when to send the next release to the client and ask for feedback without overwhelming the client.

3.6 Subsystem Decomposition

We derive ProCeeD's system design following the approach proposed by Bruegge and Dutoit [BD09]. In a first step, we decompose the objects identified in Section 3.4 into components. Then we group the components into subsystems and describe their interfaces. Figure 3.14 presents ProCeeD's components using packages as a UML Class Diagram. The ReleaseService allows the creation and promotion of releases from packaged prototypes stored in the ArtifactRepository. These prototypes are created by the BuildService component.

The BuildService retrieves PrototypeComponents either from the VersionControlRepository or a PrototypingTool. To release evolutionary prototypes, the BuildService retrieves artifacts like source code and media items from the VersionControlRepository. To allow the release of revolutionary prototypes, the PackageService first performs a set of transformation steps on the revolutionary prototype created with a PrototypingTool. The BuildService then builds and packages both kinds of prototypes. The resulting executable prototype and logs as well as metadata of the build process are stored in the ArtifactRepository.

Releases can be promoted to groups of users defined in the UserManager. The UserManager controls which user can access which release. To allow users to provide feedback to a release, two components are used: the FeedbackCollector is part of each prototype created by the BuildService and allows users of the application to provide feedback from within the application. The FeedbackRepository stores the user feedback and allows a developer to manage the received feedback items.

The MetricsManager collects the events for ProCeeD's process metrics workflow which are described in Section 3.5.2. For example, the MetricsManager can record an event whenever a group of users received a release and when the group accessed the release for the first time. Based on the tracked events, the MetricsManager computes metrics defined by a developer.

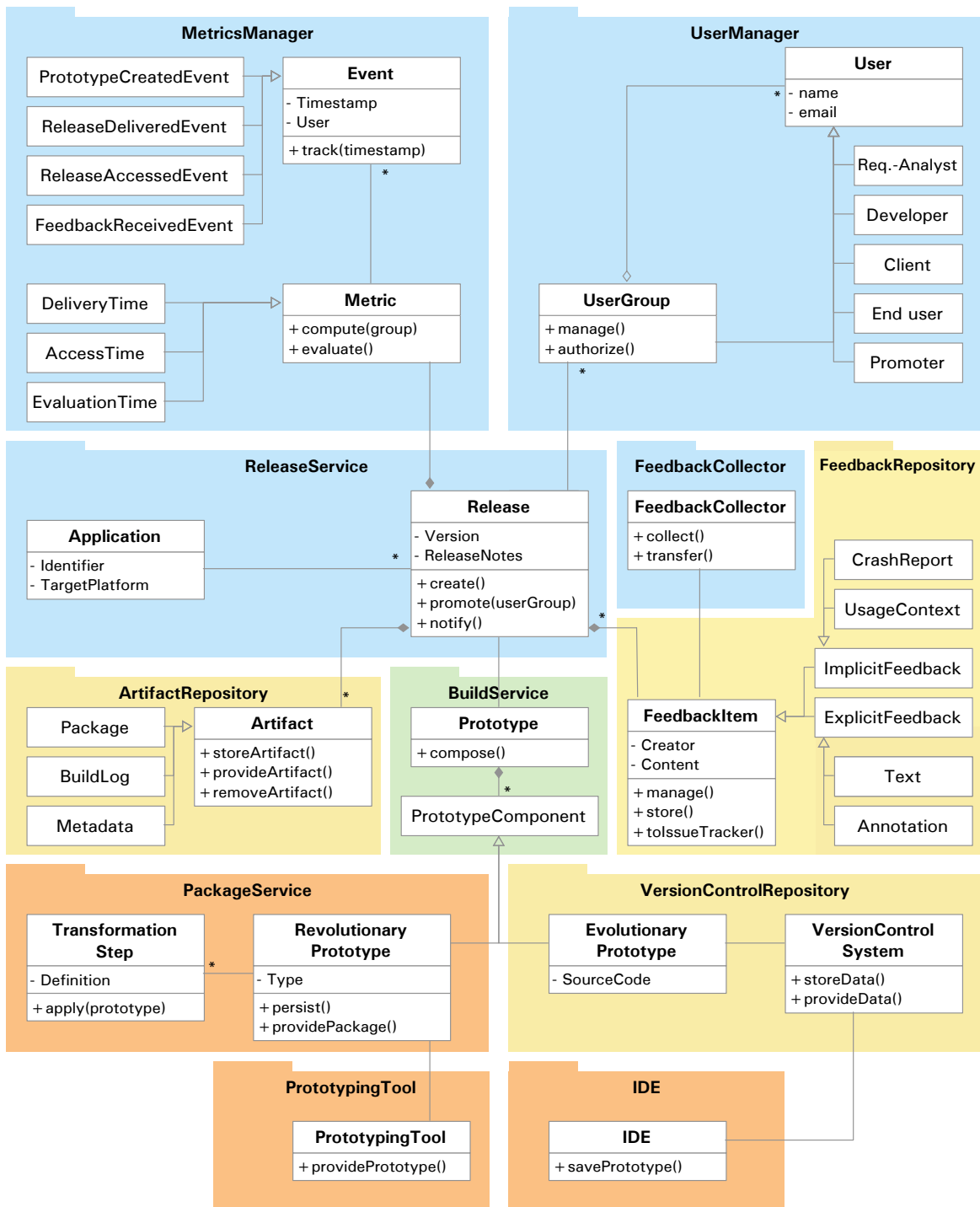


Figure 3.14: ProCeed's subsystem decomposition - objects and packages (UML Class Diagram).

Figure 3.15 shows the ProCeeD components clustered into four subsystems Development, Integration, Delivery and Repository as a UML Component Diagram. Public services which can be consumed by e.g. a graphical user interface are depicted in grey.

The Development subsystem allows developers to create and prepare prototypes for delivery to the target environment. It contains the software components to transform revolutionary prototypes created with prototyping tools into a format which can be delivered to and executed in the target environment. Evolutionary prototypes are persisted by the Repository subsystem to be later retrieved by the Integration subsystem. Revolutionary Prototypes are provided to the Integration subsystem.

The Integration subsystem is concerned with the automation of the build and package process for revolutionary as well as evolutionary prototypes. It allows developers to create executable releases from prototypes. Moreover, it stores and retrieves artifacts in the Repository subsystem. Using the Delivery subsystem, developers are able to deliver and promote release and to manage users and groups. In addition, it allows developers to monitor the delivery process using the MetricsManager. Finally, the Repository subsystem is concerned with the storage of source code, build artifacts and user feedback. Developers can manage user feedback and copy FeedbackItems to an issue tracking system.

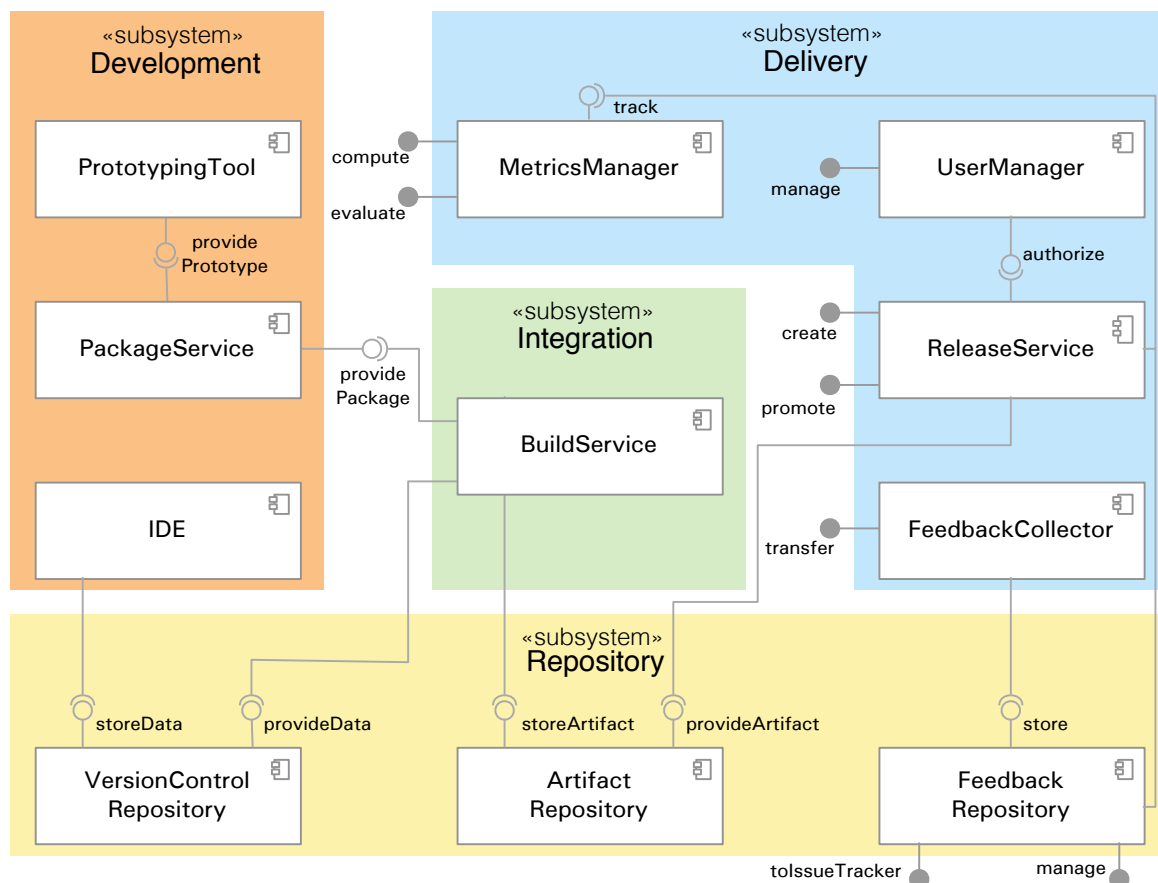


Figure 3.15: ProCeeD's subsystem decomposition - subsystems (UML Component Diagram).

3.7 Hardware/Software Mapping

In this section we present an exemplary mapping of ProCeed's subsystems and components to hardware and software nodes. The mapping is depicted in Figure 3.16 and shows a deployment of ProCeed to a private cloud environment.

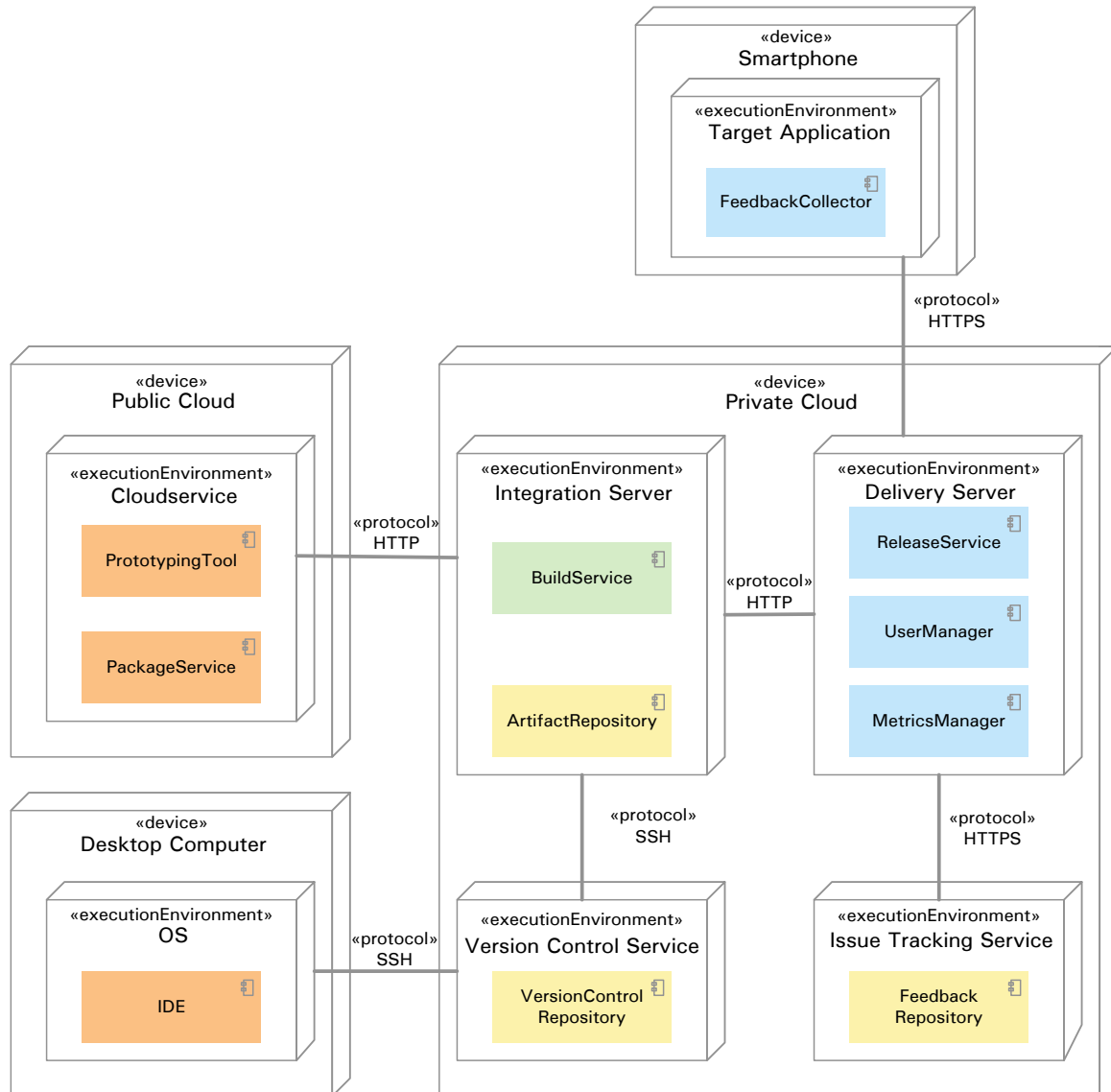


Figure 3.16: ProCeed's hardware/software mapping - example (UML Deployment Diagram).

The Integration Server accesses prototypes created in the PrototypingTool using an HTTP-based protocol. To create the source code of evolutionary prototypes, developers use an IDE running on a computer. Commercial or non-commercial software components can be used as a Version Control System, Integration Server and Issue Tracking Service. On the Integration Server node we deploy ProCeed's ReleaseService as well as the ArtifactRepository component. The node is therefore concerned with the creation and the storage of releases and their artifacts. The Delivery Server executes the PromotionManager as well as the

MetricsManager components. Both components are implemented as custom software components. The FeedbackCollector component is deployed into the target application with each release created by ProCeeD. It transfers FeedbackItems created by the user of a release to DeliveryService which stores the FeedbackItem in the FeedbackRepository of the Issue Tracking Service.

In this chapter we present Prototyper, a delivery solution for mobile applications. Prototyper uses the services and workflows offered by the ProCeed Framework and allows developers to deliver prototypes whether they were created with a prototyping tool or implemented in the programming language of the target system, as an executable application to their users. We developed Prototyper as a research instrument to evaluate the ProCeed Framework in both: a university context and in projects in industry.

The remainder of this chapter is structured as follows: Section 4.1 presents Prototyper's user interface and maps it to the ProCeed components described in Section 3.6. Section 4.2 shows the mapping of these components to hardware and software nodes and describes implementation details.

4.1 User Interface

In this section we present Prototyper's user interface model, which consists of the following elements: application management, release management, release promotion, feedback management and process metrics. For each part, we show the mapping to the respective ProCeed components, subsystems and their services. An excerpt of Prototyper's user interface showing the application management dashboard is depicted in Figure 4.1.

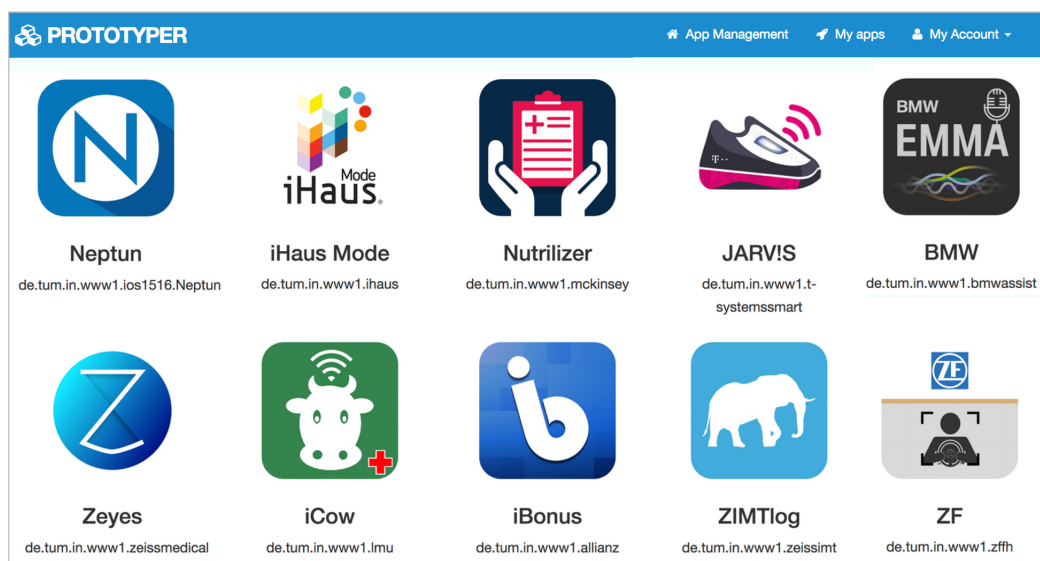


Figure 4.1: Prototyper's dashboard showing available applications.

The Dashboard presents the applications a user has access to. Each application is depicted with an icon and a unique identifier. The user interface adapts depending on the role of the current user. A developer can manage releases and feedback, while a user can execute tasks such as downloading the latest release, reading the release notes or providing feedback to a release

Release Management

Using the *Release Management UI*, developers can create releases from prototypes using ProCeed's ReleaseService. Figure 4.2 shows the involved components and services. Involved components are colored, services consumed by the release management user interface are shown in grey. When delivering a release of a revolutionary prototype created with a PrototypingTool, the PackageService transforms the prototype into an executable which can be deployed to the target environment. When creating a release from an evolutionary or hybrid prototype, ProCeed retrieves the source code of the prototype from the VersionControlRepository.

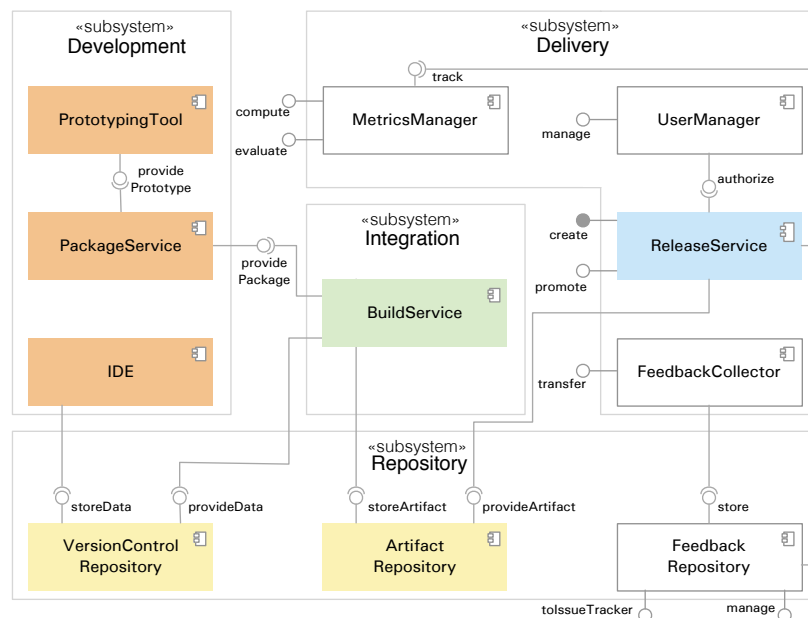
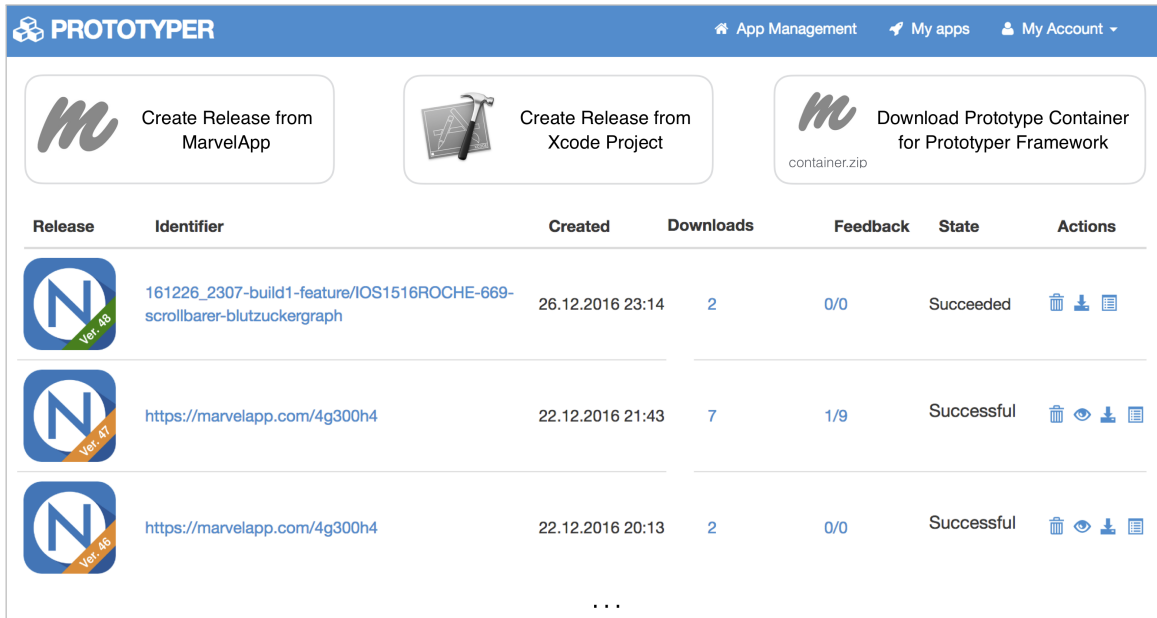


Figure 4.2: ProCeed components used for release mgmt. (UML Component Diagram).

The ReleaseService then creates an executable release for all supported prototypes and then stores the release artifacts in the ArtifactRepository. The release is now shown in the Prototyper UI and can be promoted to a group of users. Figure 4.2 illustrates an example with three releases. Releases 46 and 47 were created from a revolutionary prototype ("MarvelApp"), Release 48 from an evolutionary prototype ("Xcode Project"). By using the link in the identifier column, a developer can drill down to the underlying commit history in the *VersionControlRepository* or view the revolutionary prototype the release is based on. A developer can also track who has downloaded the release and manage associated user feedback.



The screenshot shows the Prototyper web interface. At the top, there are navigation links for 'App Management', 'My apps', and 'My Account'. Below the navigation, there are three main action buttons: 'Create Release from MarvelApp', 'Create Release from Xcode Project', and 'Download Prototype Container for Prototyper Framework'. The main content area is a table with the following columns: Release, Identifier, Created, Downloads, Feedback, State, and Actions.















Release	Identifier	Created	Downloads	Feedback	State	Actions
	161226_2307-build1-feature/IOS1516ROCHE-669-scrollbar-blutzuckergraph	26.12.2016 23:14	2	0/0	Succeeded	  
	https://marvelapp.com/4g300h4	22.12.2016 21:43	7	1/9	Successful	   
	https://marvelapp.com/4g300h4	22.12.2016 20:13	2	0/0	Successful	   

Figure 4.3: Prototyper's release management user interface showing revolutionary prototypes (orange) and evolutionary prototypes (green).

Release Promotion

The Prototyper *Release Promotion UI* allows developers to promote releases to groups of users. It builds on ProCeed's ReleaseService and UserManager components which are shown in Figure 4.4.

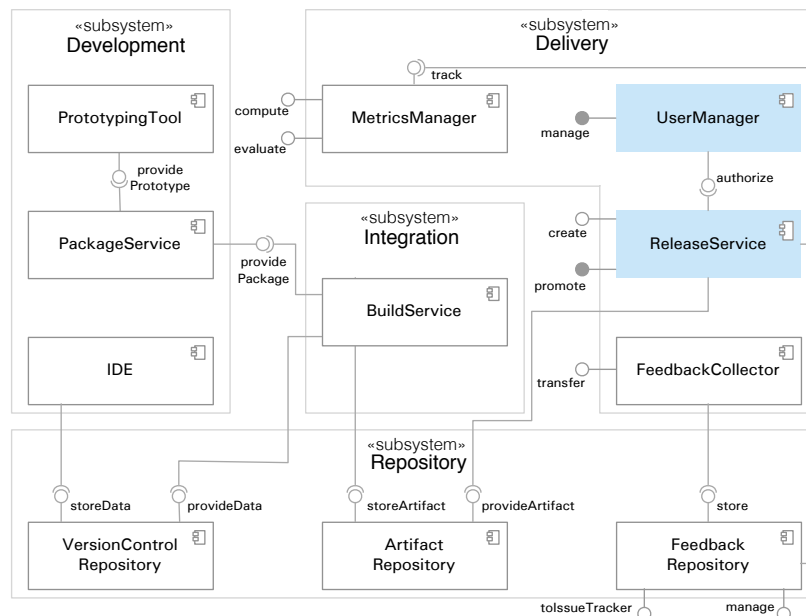


Figure 4.4: ProCeed components used for release promotion (UML Component Diagram).

A developer can promote a release to one or multiple groups of users. In addition, he can choose to deliver additional releases to a group of users at the same time. This allows developers to explore a requirement by providing multiple, alternative releases to a group of users who then can provide feedback and express which alternative they would prefer. The release promotion user interface, shown in Figure 4.5, presents developers with a release matrix which denotes what releases are currently deployed to which user group. Figure 4.5 depicts an example with three user groups. While the developers are running the newest release 5, clients and users are currently on release 4 and can access an additional, revolutionary prototype, marked as release 3.

The screenshot shows the Prototyper interface for the 'Diabetes App'. At the top, there's a navigation bar with 'App Management', 'My apps', and 'My Account'. Below that, the app details are shown: 'Diabetes App' with a URL 'de.tum.in.www1.diabetesapp' and a creation time of 'Created about 7 hours ago'. The main section is titled 'Release groups' and contains a matrix with columns for 'Developers', 'Clients', and 'Users', and rows for 'Release' and 'Additional Release'. The 'Release' row shows 'Rel. 5' for Developers, 'Rel. 4' for Clients, and 'Rel. 4' for Users. The 'Additional Release' row shows a placeholder icon for Developers, 'Rel. 3' for Clients, and 'Rel. 3' for Users. Below this is a 'Releases' table with columns for 'Release', 'Identifier', 'Downloads', 'Feedback', 'State', and 'Action'. The table contains one entry for 'Rel. 5' with identifier 'IOS1516ROCHE-IOS1516ROCHE', 0 downloads, 0 feedback, and a 'Succeeded' state.

Release groups	Developers	Clients	Users
Release	Rel. 5	Rel. 4	Rel. 4
Additional Release		Rel. 3	Rel. 3

Release	Identifier	Downloads	Feedback	State	Action
Rel. 5	IOS1516ROCHE-IOS1516ROCHE	0	0	Succeeded	

Figure 4.5: Prototyper's release promotion user interface.

A developer can design a promotion workflow tailored to the needs of the project he is working on. Before delivering a release to a client he could e.g. define a group of developers executing manual user interface tests. When promoting the release, developers can provide release notes using the *Release Notes UI* depicted in Figure 4.6. The release notes are shown to each member of the according user group. Prototyper allows developers to refine release notes on each promotion step (1). While an internal release to fellow developers may only contain brief comments on the last changes, a developer can refine these release notes before the next promotion step (2).

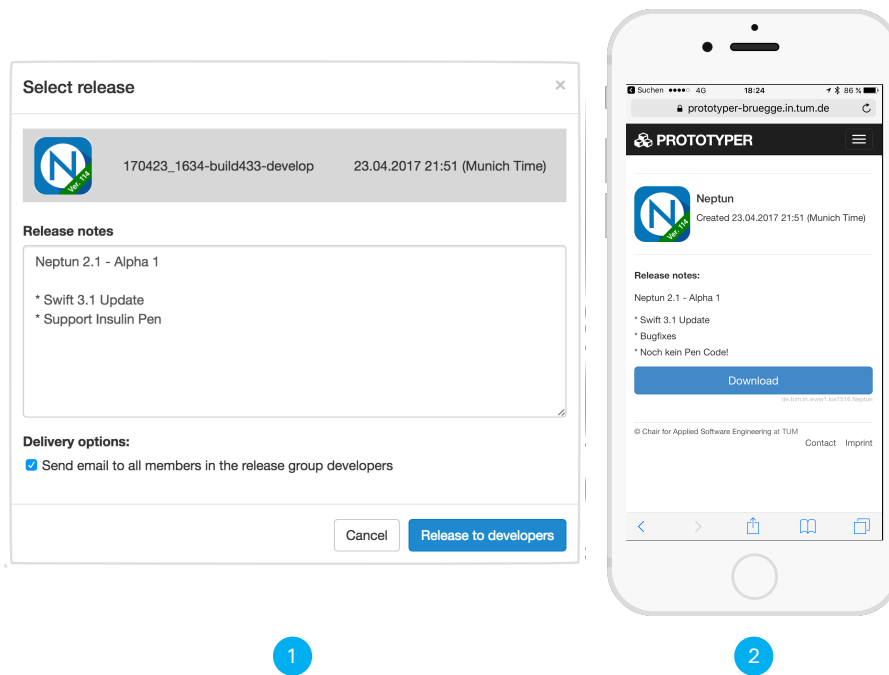


Figure 4.6: Prototyper's release notes user interface.

Feedback Management

Prototyper's *User Feedback UI* is split into two parts which are based on the components depicted in Figure 4.7. The FeedbackRepository allows developers to manage feedback received by users. A developer can read the textual feedback and look up annotated screenshots.

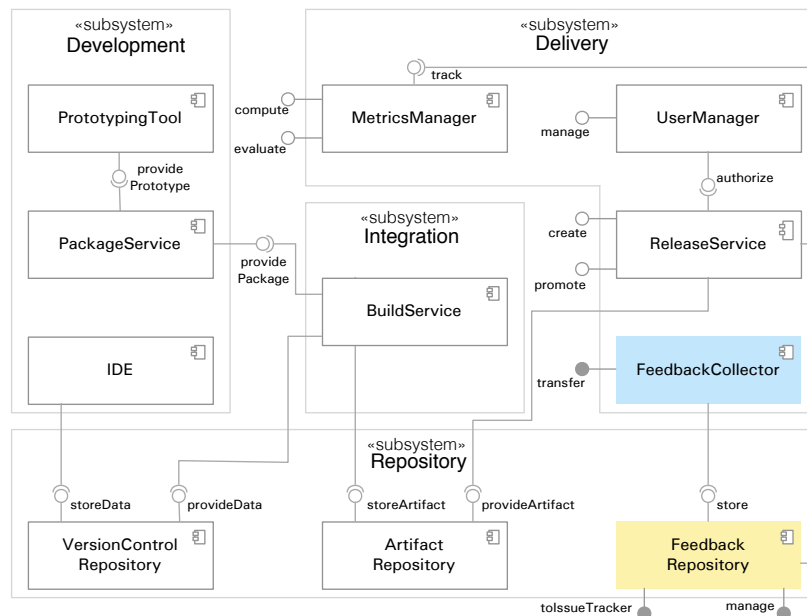
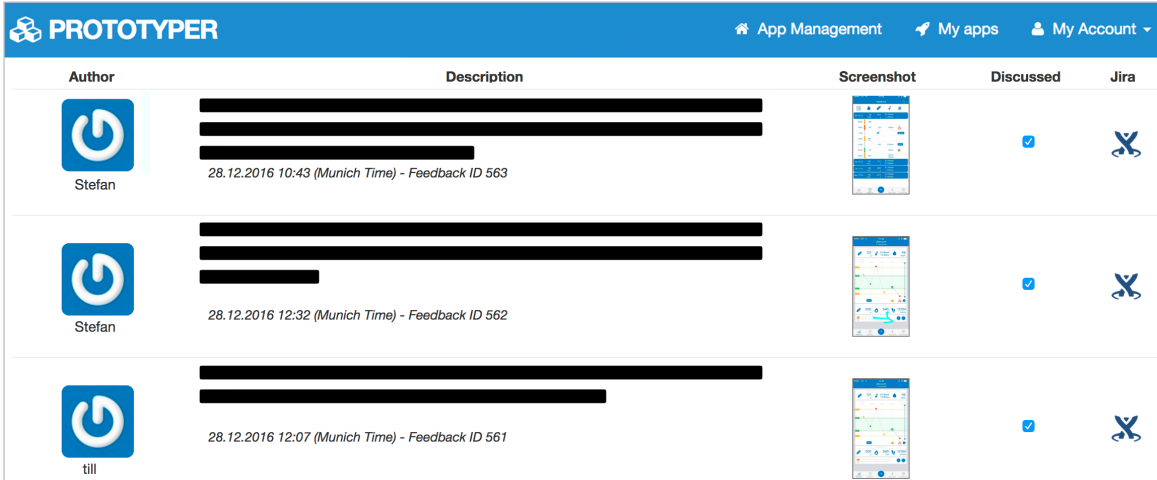


Figure 4.7: ProCeed components used for feedback mgmt. (UML Component Diagram).

To further process feedback items, he can decide to transfer items to an issue tracking system. Figure 4.8 depicts an example with three feedback items. Each shows the author, their textual feedback and, if provided by the user, an annotated screenshot.



The screenshot shows the Prototyper web interface. At the top is a blue header with the Prototyper logo and navigation links for 'App Management', 'My apps', and 'My Account'. Below the header is a table with five columns: 'Author', 'Description', 'Screenshot', 'Discussed', and 'Jira'. There are three rows of feedback items. Each row shows a user profile picture and name, a redacted description, a mobile app screenshot, a 'Discussed' checkbox (all checked), and a 'Jira' icon.










Author	Description	Screenshot	Discussed	Jira
 Stefan	[Redacted] 28.12.2016 10:43 (Munich Time) - Feedback ID 563		<input checked="" type="checkbox"/>	
 Stefan	[Redacted] 28.12.2016 12:32 (Munich Time) - Feedback ID 562		<input checked="" type="checkbox"/>	
 till	[Redacted] 28.12.2016 12:07 (Munich Time) - Feedback ID 561		<input checked="" type="checkbox"/>	

Figure 4.8: Prototyper’s feedback management user interface.

The second part of Prototyper’s user feedback interface builds on the FeedbackCollector component and is deployed with each release. Using the feedback user interface shown in Figure 4.9, users can provide in-situ feedback while evaluating a release in the target environment. While using the application, a user can tap the blue feedback button (1). They can then add an annotated screenshot (2, 3) to a feedback item and provide textual feedback (4).

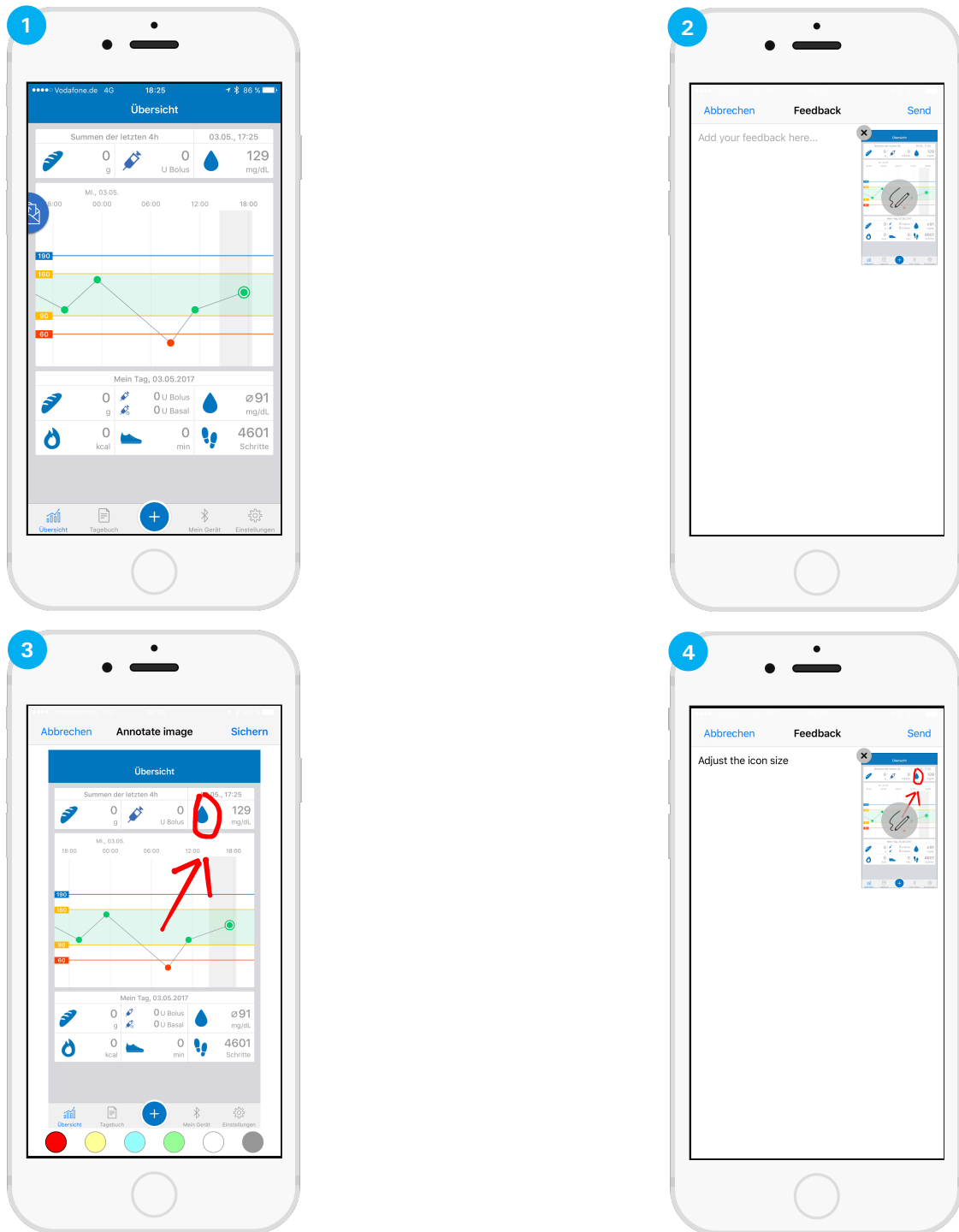


Figure 4.9: Prototyper's FeedbackCollector component embedded in a mobile application.

Process Metrics

Following ProCeed’s process metrics workflow, Prototyper tracks the events modeled in ProCeed’s object model presented in Section 3.4. Building on top of ProCeed MetricsManager component, depicted in Figure 4.10, Prototyper collects events generated by ProCeed’s ReleaseService and FeedbackCollector.

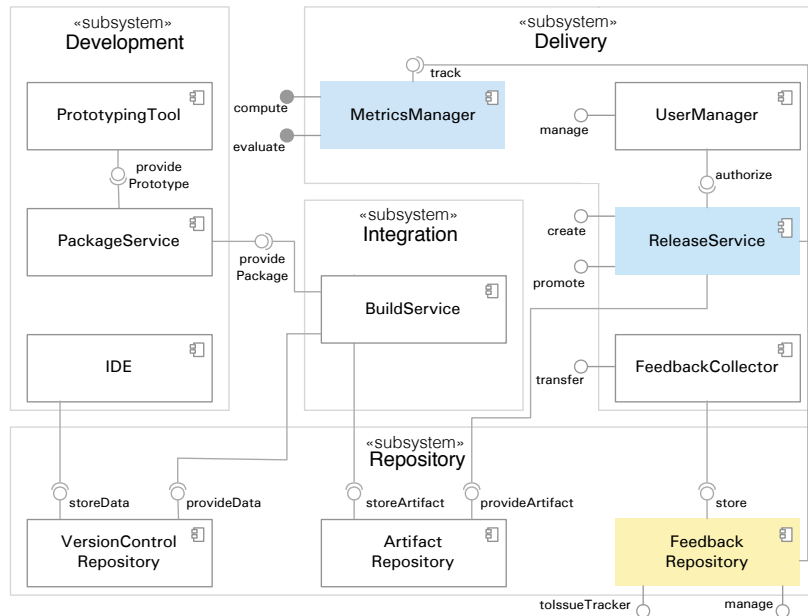


Figure 4.10: ProCeed components used for process metrics (UML Component Diagram).

Based on the collected events, developers can define and monitor the metrics described in Section 3.4 for each release.

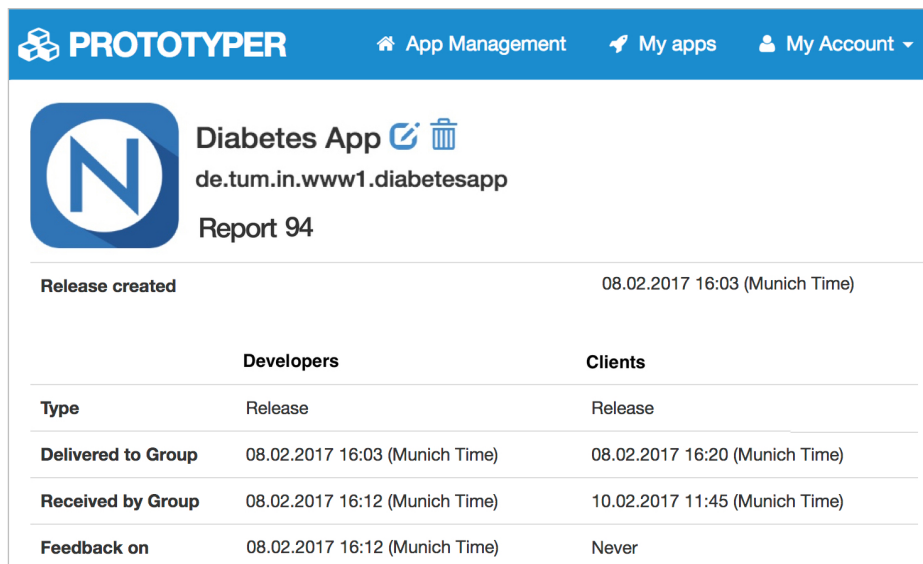


Figure 4.11: Prototyper’s process metrics user interface showing metrics for a release.

Metrics are calculated on a per-user-group-basis. Developers can download reports visualizing the release process of the project and use these reports e.g. to improve the team's release workflow or to decide when to send the next release to a group of users. Figure 4.11 shows an exemplary report for a release which has been delivered to two user groups. A developer has already provided feedback, while feedback from the client is still missing.

4.2 Hardware/Software Mapping

Figure 4.12 presents the mapping of Prototyper and ProCeeD components to actual hardware and software as well as protocols. For the scope of this research, we deployed several ProCeeD components using the Atlassian¹ software development stack. We used Atlassian's Bitbucket Server² as a VersionControlRepository, their Bamboo³ integration service for the BuildService as well as the ArtifactRepository and the issue tracker JIRA⁴ as a FeedbackRepository.

As a PrototypingTool we used MarvelApp⁵, a web-based solution for the creation of user interface prototypes like wireframes and digital prototypes. MarvelApp also provides a smartphone app which allows developers to digitize prototypes created on paper.

Components we implemented on our own are marked in blue. With Prototyper's PackageService on the Prototyper Package node we implemented a component to transform prototypes created using MarvelApp into mobile iOS Applications which can be build and code signed by the BuildService. The Prototyper Delivery node executes the ProCeeD ReleaseService, UserManager and MetricsManager components. In addition, Prototyper Delivery executes the FeedbackRepository which can be synchronized with Atlassian JIRA by a developer. The FeedbackCollector component was implemented as an iOS framework which is integrated into each revolutionary and evolutionary prototype delivered using Prototyper.

In the following sections we describe how developers can deliver revolutionary as well as hybrid prototypes using the nodes Prototyper Package containing the PackageService, Application containing the FeedbackCollector component and Prototyper Delivery executing the UserManager, MetricsManager and FeedbackRepository components.

¹<http://www.atlassian.com>

²<http://www.atlassian.com/bitbucket>

³<http://www.atlassian.com/bamboo>

⁴<http://www.atlassian.com/jira>

⁵<http://www.marvelapp.com>

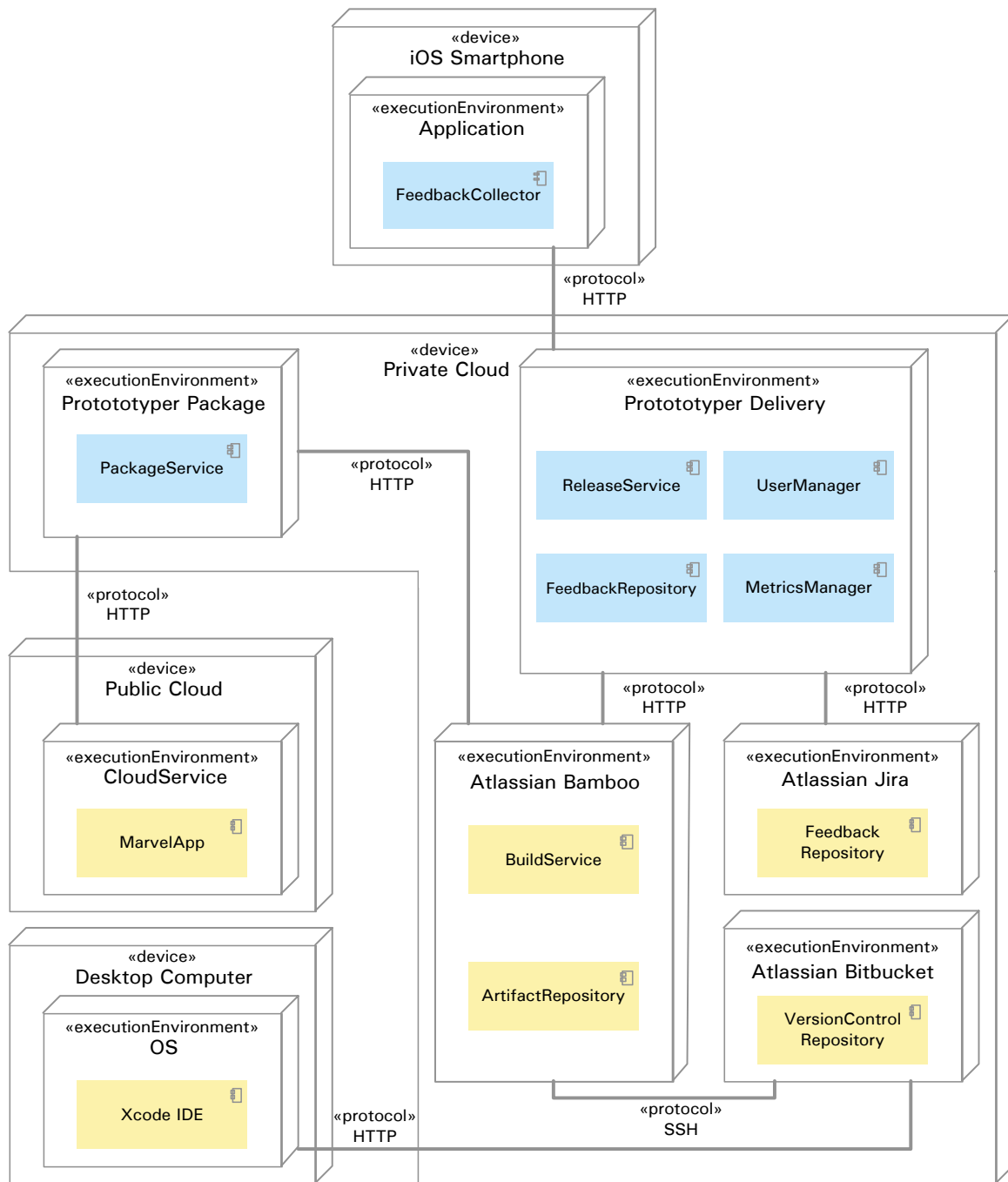


Figure 4.12: Prototyper’s hardware/software mapping (UML Deployment Diagram). Components implemented as part of Prototyper are drawn blue, components based on COTS software are drawn yellow.

Create a Release of a Revolutionary Prototype

Developers can use the PackageService to transform revolutionary prototypes created using a prototyping tool into a mobile applications which can be executed in the target environment. The workflow is as follows: The PackageService downloads a representation of the revolutionary prototype from the prototyping tool MarvelApp. It supports various input formats, e.g. PDF, a series of images, web-based representations or structured formats like XML or JSON. Using pre-defined transformation steps for the corresponding prototyping tool, the PackageService transforms the revolutionary prototype into an intermediate format which is then embedded in a template application, e.g. a mobile iOS or Android application. In addition, Prototyper Package adds the FeedbackCollector component for in-situ user feedback to the application.

After the prototype package is finished, the ReleaseService builds, packages and code-signs the revolutionary prototype as a mobile application using the integration service Atlassian Bamboo. The release is now ready to be delivered to a group of users. If an error occurs, the developer is informed.

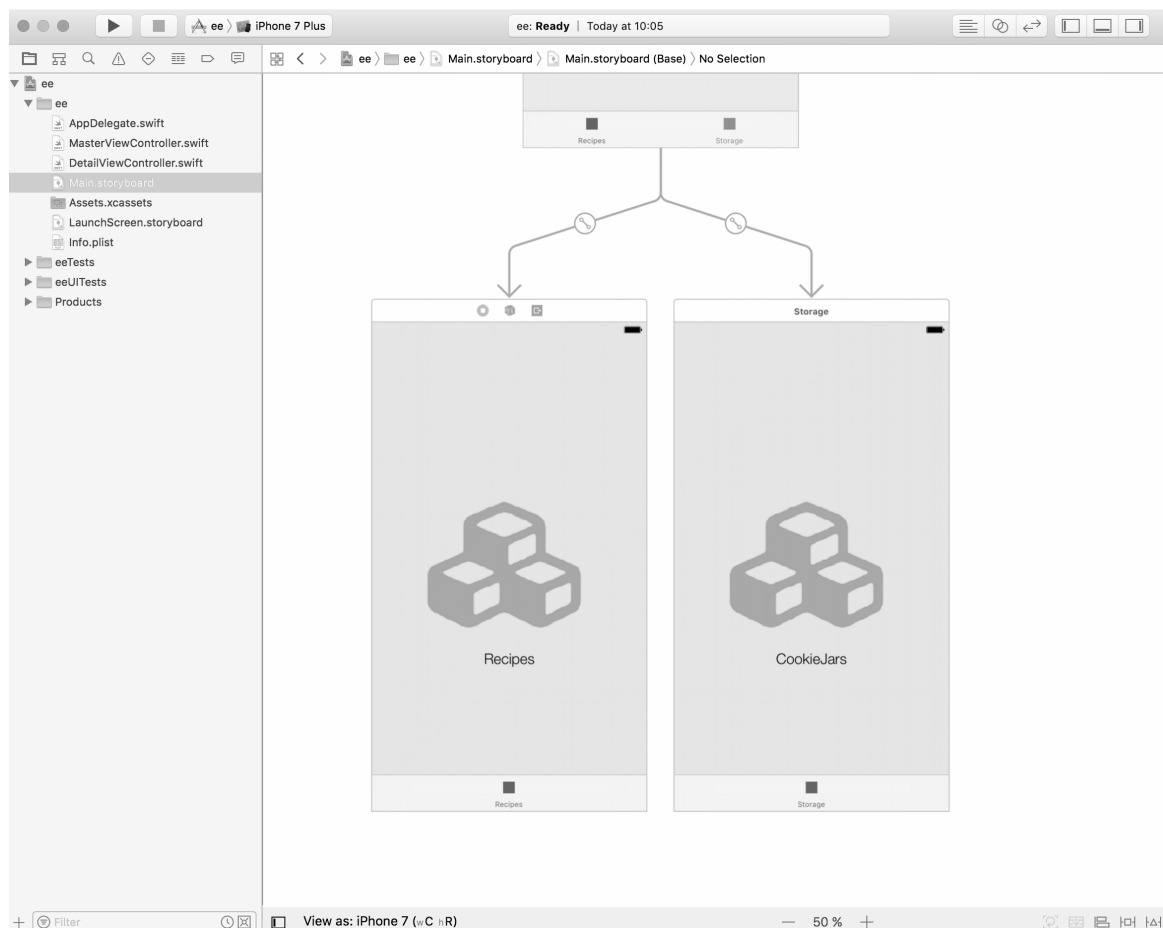


Figure 4.13: Revolutionary and evolutionary prototypes combined into a hybrid prototype.

Create a Release of a Hybrid Prototype

Prototyper Package allows developers to create hybrid prototypes consisting of an application already written in the programming language of the target system and parts of revolutionary prototypes. Thus, the developer can integrate parts of a revolutionary prototype into to an already existing application to e.g. show a feature which is only partially implemented.

The workflow is as follows: Using the PackageService a developer creates a representation of the revolutionary prototype he wants to integrate. The developer then downloads the representation and integrates it into his app project using the IDE. He then integrates the prototype into the app using a lightweight integration framework. The framework parses the packaged prototype and allows the developer to replace e.g. one or multiple screens with screens of the prototype. Figure 4.13 shows an example in which a developer has integrated a MarvelApp prototype into an existing mobile application using the Xcode IDE⁶. The developer composed the user interface shown with parts of a revolutionary prototype ("Recipes", "CookieJars").

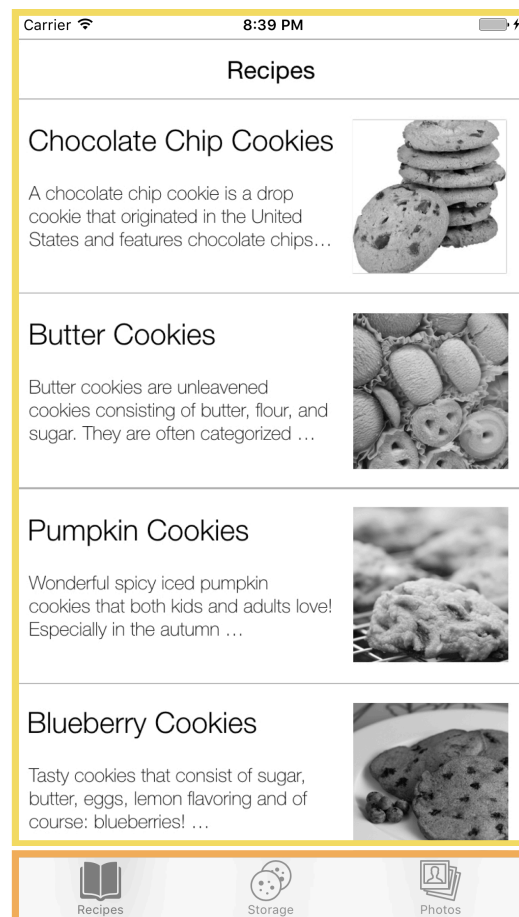


Figure 4.14: Mobile app created using Prototyper. Yellow parts are based on a revolutionary prototype, orange parts are implemented in code.

⁶<http://www.apple.com/Xcode>

After this step, a release containing both parts, i.e. the parts of the application which are written in code and the revolutionary prototype, are delivered and can be evaluated in the target environment. Figure 4.14 shows the example app which has been created using this hybrid approach. It consists of two parts: The developer has combined parts of a revolutionary prototype (yellow) with an existing evolutionary prototype, developed in the programming language of the target app (orange).

Deliver a Release

The ReleaseService allows the delivery of revolutionary, evolutionary and hybrid prototypes using a common workflow, interaction design and technological foundation. If a developer wants to deliver multiple design alternatives of a prototype to a group of testers in order to get feedback and decide which one to choose, Prototyper automates all necessary deployment steps. A user can download and access each release simultaneously on their mobile device and compare them in the target environment.

As defined in the ProCeeD Framework, Prototyper also implements a common in-app feedback system for revolutionary as well as evolutionary prototypes. The system is implemented using two components: the FeedbackCollector component is deployed into each release delivered with Prototyper. It allows users to provide in-app feedback while using the application. The FeedbackRepository component is part of Prototyper and allows developers to manage and further process user feedback. While Prototyper implements a FeedbackRepository component, feedback can also be stored in an issue tracking system for further triage or implementation.

In the previous chapters we described the ProCeeD Framework and Prototyper as a tool building upon it. This chapter presents the empirical evaluation of Prototyper and ProCeeD in four case studies.

Section 5.1 describes our hypotheses and the research methodology applied. Section 5.2 presents the quantitative and qualitative evaluation of ProCeeD's delivery automation and requirements exploration workflow in a multi-project university capstone course with more than 200 developers working in 22 teams. Section 5.3 evaluates the application of ProCeeD's process metrics workflows in a project course running over multiple semesters. Section 5.4 describes the application of Prototyper and ProCeeD in a commercial project with a partner from the pharmaceutical industry. Section 5.5 presents how ProCeeD can be applied in projects using a storyboard-based requirements elicitation approach. Section 5.6 summarizes our findings. Section 5.7 discusses possible threats to validity.

5.1 Overview

In this section we present our hypotheses as well as the research methodology applied. While ProCeeD can be adopted for the development of any interactive software system, this evaluation focuses on the field of mobile application engineering. Mobile applications are interactive systems and may involve multiple input and output modalities. User interface prototyping is applied during the development of such systems. All four case studies cover projects concerned with mobile system development for the Apple iOS ecosystem. While case studies I, II and IV were conducted in an academic context with clients from industry, case study III is based on an industry project.

We realized the functional requirements of ProCeeD using the Prototyper tool in Chapter 4. Based on the functional and nonfunctional requirements defined in Section 3.3.2 this evaluation assesses the following aspects of ProCeeD: the unified delivery process for revolutionary and evolutionary prototypes; process improvements in the field of prototype delivery; the process adoption and the traceability of the delivery process.

5.1.1 Hypotheses

In the following we present the five hypotheses evaluated in this dissertation.

Delivery Process

To evaluate the concept of a common delivery process, we hypothesize that ProCeeD allows developers to apply a common delivery process for revolutionary and evolutionary prototypes delivered during a software project.

H1: With ProCeeD developers adopt a common delivery process for revolutionary as well as evolutionary prototypes.

H1a Using ProCeeD developers deliver revolutionary prototypes using an automated process.

H1b Using ProCeeD developers deliver revolutionary prototypes using the same tools and delivery process as they use for evolutionary prototypes.

Process Improvements

To investigate how ProCeeD improves the delivery of prototypes, we hypothesize that ProCeeD improves the prototype delivery process by lowering the amount of time necessary to deliver a prototype and shortening the length of a prototyping iteration compared to teams not adopting ProCeeD.

H2: ProCeeD reduces the length of a prototyping iteration.

H2a Using ProCeeD developers need less time to deliver prototypes compared to developers not using ProCeeD.

H2b ProCeeD teams see a shorter overall iteration length compared to Non-ProCeeD teams.

We further hypothesize that ProCeeD improves user involvement during requirements engineering by allowing developers to collect more and better feedback compared to projects not adopting by ProCeeD.

H3: ProCeeD allows developers to collect more and better feedback.

H3a Teams who use ProCeeD see more feedback than Non-ProCeeD teams.

H3b Teams who adopt ProCeeD receive better feedback than Non-ProCeeD teams.

Process Adoption

To evaluate ProCeeD's adoption by developers, we hypothesize that developers see a benefit for their development workflow when applying ProCeeD.

H4: Developers see a benefit of applying ProCeeD in their workflow.

H4a Developers use ProCeeD to download a release.

H4b Developers use ProCeeD to lookup feedback.

H4c Developers use ProCeeD's integration with the issue tracking tool and communication tools.

Process Traceability

To assess how ProCeeD improves the traceability of the delivery process we hypothesize that ProCeeD reduces the amount of time developers need to monitor the delivery process of multiple project teams. We further hypothesize that developers utilize ProCeeD to decide when to interact with their users by e.g. sending a new release or asking their clients for feedback.

H5: ProCeeD improves the traceability of the software delivery process in multi-project organizations.

H5a ProCeeD reduces the time needed to monitor the delivery process.

H5b ProCeeD allows developers to optimize the frequency of their deliveries.

5.1.2 Methodology

To evaluate the above-described hypotheses we conducted four case studies. An overview of the case studies mapped to the ProCeeD workflows evaluated is depicted in Figure 5.1.

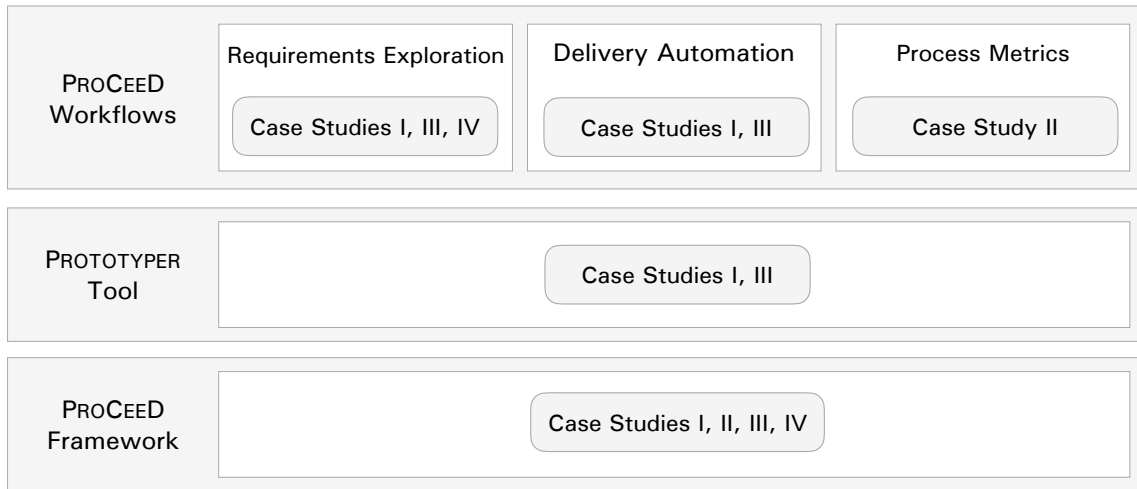


Figure 5.1: Mapping of the case studies to the ProCeeD workflows.

Case study I evaluates Prototyper and ProCeeD’s delivery automation and requirements exploration workflows in two instances of a multi-project capstone course with more than 200 developers working in 22 projects with clients from industry. In the case study, presented in Section 5.2, we quantitatively evaluated ProCeeD over the course of one year. To complement our findings, we present the results of a qualitative user survey we conducted with 11 project teams. *Case study II*, presented in Section 5.3, qualitatively evaluates ProCeeD’s process metrics workflow in a project-based capstone course across multiple semesters. *Case study III* qualitatively evaluates the use of Prototyper and ProCeeD in a commercial project in the pharmaceutical industry over the course of 18 months. The study is described in Section 5.4. *Case study IV* introduces ProCeeD in an innovation project using a storyboard-based requirements elicitation approach. The case study follows a qualitative research approach and is presented in Section 5.5.

5.2 Case Study I: University Capstone Course

In this confirmatory case study [ESSD08], we describe the adoption and usage of ProCeeD's delivery automation and requirements exploration workflows in a multi-project capstone course with clients from industry. We quantitatively evaluated the delivery process of all participating teams and conducted a qualitative evaluation based on an online questionnaire afterwards. The study design is based on a pre-study we published in [Alp+17]. Section 5.2.1 describes the study design; it includes the participants, environment and our research approach. Section 5.2.2 and 5.2.3 report the quantitative and qualitative study results. Section 5.2.4 discusses our findings.

5.2.1 Design

This section presents the participants of the study and describes the study environment.

Participants

Participants of the capstone course typically have a background in computer science and a varying levels of experience. Table 5.1 presents the participants of the two instances of the course evaluated in this case study. In total, 159 students working in 22 teams of 6-8 developers took part. Each team was lead by a doctoral student in the role of the project leader and an experienced student in the role of a coach. 11 teams ("ProCeeD teams") were using ProCeeD and Prototyper to deliver revolutionary as well as evolutionary prototypes. 11 teams ("Non-ProCeeD teams") delivered revolutionary prototypes using a method of their choice and a delivery process based on Atlassian Bamboo¹ and Microsoft HockeyApp² for evolutionary prototypes [KA14]. The Non-ProCeeD teams took part during the winter semester 2015/2016, the ProCeeD teams during the summer semester 2016.

Table 5.1: Case study I: Participants

Teams	Participants	Coaches	Project Leaders	Team-Size
Non-ProCeeD	79	11	11	6-8 (+ 2)
ProCeeD	80	11	11	6-8 (+ 2)

Environment

The instances of the course took part during the winter semester 2015/2016 and summer semester 2016 respectively. During the course, participating students worked in teams over the period of three months to develop mobile applications and the corresponding systems with partners from industry. The course is based on the Rugby process model [BKA15], an agile development model in the context of Continuous Software Engineering which is

¹<http://www.atlassian.com/bamboo>

²<http://www.hockeyapp.net>

based on Scrum. Rugby is described in detail in [KABW14], [KA14] and [Kru16]. Figure 5.2 shows the organizational structure of the course. Two instructors are responsible for the overall management. Each project team is composed of one or many clients from industry, a project leader, a coach as well as the students who are in the role of developers.

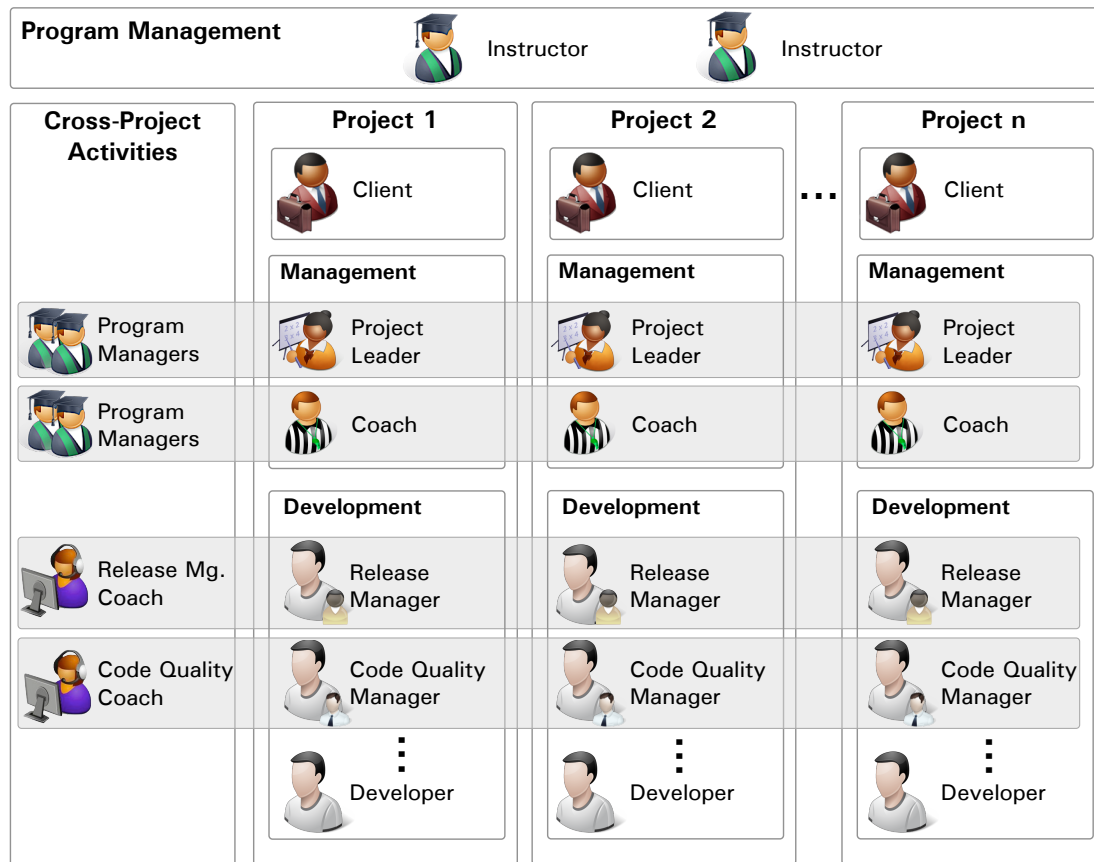


Figure 5.2: Case study I: Organizational chart of the course (adapted from [BKA15]).

The *client* is an employee of a partner from industry. The *project leader* is a teaching assistant in the role of a scrum master. The *coach* supports a project leader in his daily work. Some of the developers take over the responsibility for additional topics in the team, e.g. release management. At the beginning of the course, the clients present their projects ideas to the students in a Kickoff meeting. Afterwards students are allocated into teams of 6-8 developers. Teams develop their projects in sprints of one to two weeks. For each sprint planning and review meeting the client usually meets with the team in person. In the *Design Review* meeting, around 7 weeks after the Kickoff, each team presents the current state of their project including a detailed system design. At the end of the course and during the *Client Acceptance Test* each team presents their system in front of all participants and clients. We describe the course organization in detail in [BKA15].

Research Process

The study was conducted as follows: all teams, ProCeeD teams as well as Non-ProCeeD teams, were taught the basics of prototyping and release management in interactive tutorials given by the course instructors at the beginning of the course. The ProCeeD teams were introduced to Continuous Prototyping and the Prototyper tool in a separate, in-class tutorial where the course instructors presented them with the Continuous Prototyping approach described in this dissertation. An overview of the prototyping process applied by the teams is shown in Figure 5.3.

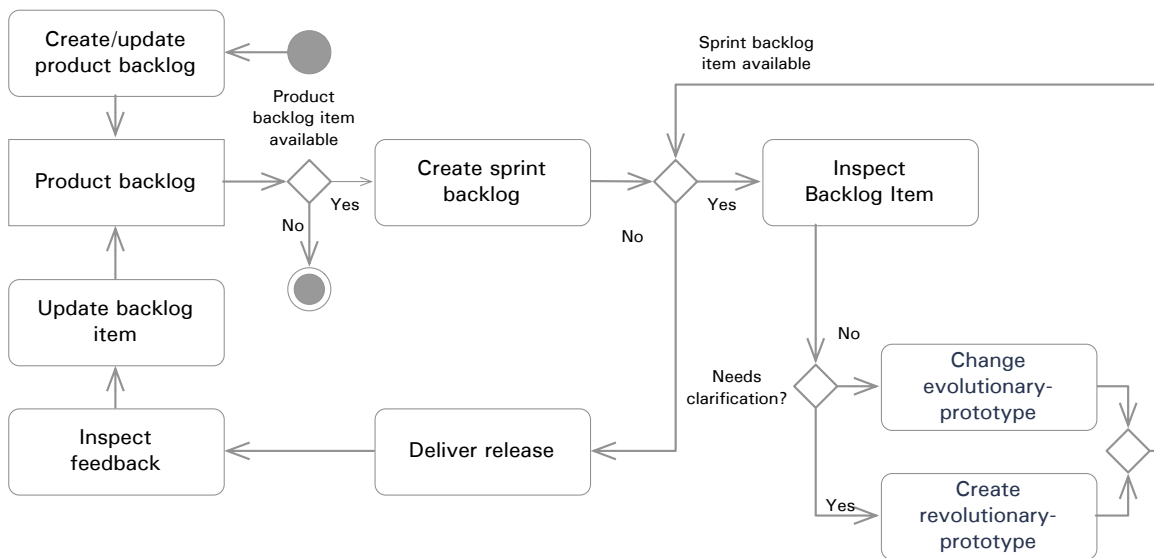


Figure 5.3: Case study I: Continuous Prototyping process applied.

ProCeeD teams were asked to decide for each backlog item if the underlying requirement is already clear enough to be implemented during the next sprint or if the backlog item needs further refinement and discussion, in which case the team could decide to create and deliver a prototype to do so. Teams were asked to not only deliver a product increment at the end of a sprint but to deliver prototypes to their clients any time they needed feedback.

During the course, we asked the ProCeeD as well as Non-ProCeeD teams to gather feedback from their client not only in personal meetings but using any means of communication. For the ProCeeD teams we configured Prototyper to include the FeedbackCollector component, described in Section 3.6, in each delivered revolutionary prototype in order to allow users to give textual feedback or to create and annotate screenshots while evaluating the prototype on their mobile device. We also asked the teams to include the FeedbackCollector component in each delivery of an evolutionary prototype they created using their IDE. The Non-ProCeeD teams did not use Prototyper to deliver prototypes to their client, but used a delivery method of their choice to deliver revolutionary prototypes, e.g. email or personal meetings. For the delivery of evolutionary prototypes they used the continuous delivery process described in [KA14].

During the course we continuously monitored the usage of Prototyper and the adoption of the ProCeeD workflows. Moreover, we held weekly meetings with the release managers of both ProCeeD teams and Non-ProCeeD teams and helped them with any issues they faced while adopting the delivery process in their team.

In addition we calculated several metrics regarding the delivery process of each team based on ProCeeD's process metrics workflow we described in Section 3.3.1. The metrics delivery time, feedback time and iteration length and the underlying events (depicted in Figure 5.4) are defined as follows:

- **Delivery time:** Interval between a team completing a prototype ("Prototype CREATED") and delivering it to the client ("Prototype DELIVERED").
- **Feedback time:** Interval between delivering a prototype to the client ("Prototype DELIVERED") and receiving feedback from the client ("Feedback RECEIVED").
- **Iteration length:** Interval between a team completing a prototype ("Prototype CREATED") and receiving feedback from the client ("Feedback RECEIVED")

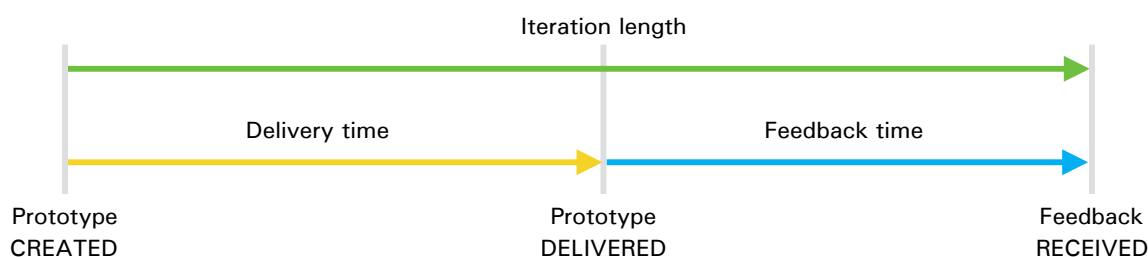


Figure 5.4: Case study I: Tracked events (capital letters) and calculated metrics (arrows).

We calculated the metrics for each release the teams sent to their clients. Releases delivered to other user groups than the client, e.g. to the development team, were not taken into account in this case study. The events `PrototypeCreated`, `PrototypeDelivered` and `FeedbackReceived` were collected as follows: For the ProCeeD teams we used ProCeeD's process metrics functionality. In addition, we manually added the `FeedbackReceived` event using spreadsheets filled out by the team to cover situations in which clients used feedback channels ProCeeD did not track automatically, e.g. feedback given during personal meetings. For the Non-ProCeeD teams we collected the events using spreadsheets which were filled out by the teams and the course instructors on a weekly basis.

After the end of the capstone course we asked all members of the ProCeeD teams to fill out a questionnaire about their experiences with ProCeeD and Prototyper. The answers to the questionnaire were collected anonymously and participation was voluntary. The participants had two weeks to complete the online questionnaire, and a reminder was sent to those who did not complete it within one week. We invited 102 participants to take the survey and received 76 valid responses, which amounts to a response rate of 74.5%.

An overview of the questionnaire is shown in Table 5.2. In question group A we asked about the participants' prior knowledge and the mobile devices they used during the course. In question group B we asked the participants about the tasks they executed using Prototyper and feedback channels used by their client. Question group C covered the adoption and usage of the ProCeeD workflows requirements exploration and delivery automation. In question group D we inquired about the different integrations Prototyper offers with development tools as well about their overall opinion about Prototyper and their ideas for future improvements.

Table 5.2: Case study I: Questionnaire - overview

Group	No	Type	Description
A	Prior knowledge and background		
	1	5-Likert (5 items)	Prior Knowledge
	2	Yes/No	Devices Use
B	Tasks and Feedback		
	1	Single Choice (6 items)	Frequency of tasks executed by participant using Prototyper
	2	Single Choice (6 items)	Frequency of tasks executed by team using Prototyper
	3	Multiple Choice	Communication channels used for a new release
	4	Text	Release workflow as applied in the team
	5	Multiple Choice	Feedback channels used by the client for revolutionary prototypes
	6	Multiple Choice	Feedback channels used by the client for evolutionary prototypes
C	Workflows		
	1	5-Likert (6 items)	Workflow - Requirements Exploration
	2	Text	Workflow - Requirements Exploration - Application Example
	3	5-Likert (6 items)	Workflow - Delivery Automation
	4	Text	Remarks Workflows
D	Prototyper Tools		
	1	5-Likert	Integration Prototyper with development tools (5 items)
	2	Text	Prototyper - Pros
	3	Text	Prototyper - Cons
	4	Text	Prototyper - Improvements

5.2.2 Quantitative Results

In this section we present the quantitative results of the case study. We analyzed the following amount of releases. For the ProCeeD teams, 140 releases were delivered to the client, 96 of which received feedback (68%). For the Non-ProCeeD teams 69 of the 114 total releases delivered to the client received feedback (60%).

Amount of Releases by Type

We first analyzed the amount of releases delivered by ProCeeD teams and Non-ProCeeD teams. Figure 5.5 shows the total amount of deliveries split by release type. While Non-ProCeeD teams delivered a total amount of 21 revolutionary prototypes to their clients, ProCeeD teams delivered a total of 43 revolutionary prototypes; 4 of them were delivered manually, e.g via paper or email, and 39 of them were delivered using the automated process of the Prototyper tool.

Independently from the delivery method, ProCeeD teams delivered 105% more revolutionary prototypes than Non-ProCeeD teams, 90% of them using Prototyper. With regard to the amount of evolutionary prototypes delivered, Non-ProCeeD teams delivered 97 releases compared to 93 releases delivered by the ProCeeD teams.

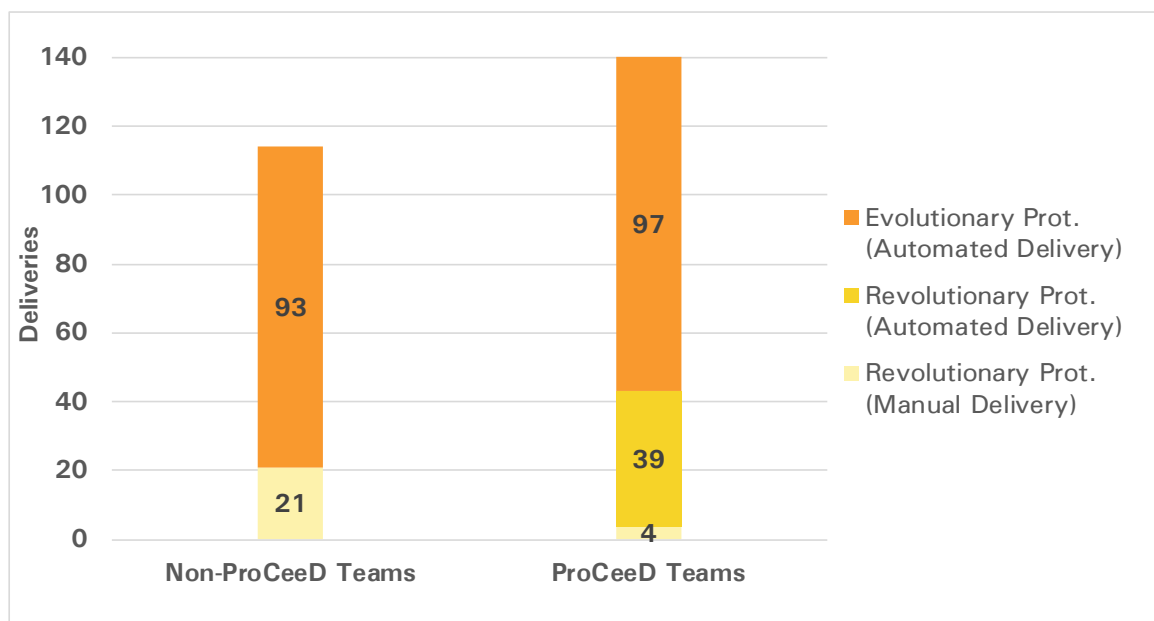


Figure 5.5: Case study I: Amount of prototypes delivered to the client.

Frequency of Releases

Figure 5.6 shows all automated deliveries to the client carried out by ProCeeD and Non-ProCeeD teams on a weekly basis. As the instance of the capstone course in which we introduced ProCeeD and Prototyper was conducted during the winter semester, we omitted the two weeks semester break to increase the comparability of the two instances of the course. Manual deliveries, e.g. via paper or email, are not included. During weeks 1-6 of the course, ProCeeD teams showed a 2-fold increase in automated deliveries compared to Non-ProCeeD teams. During weeks 7-13, the number of automated deliveries of ProCeeD and Non-ProCeeD teams were close to each other, with ProCeeD teams showing more deliveries in week 7 to 10 and Non-ProCeeD teams in weeks 11 to 13.

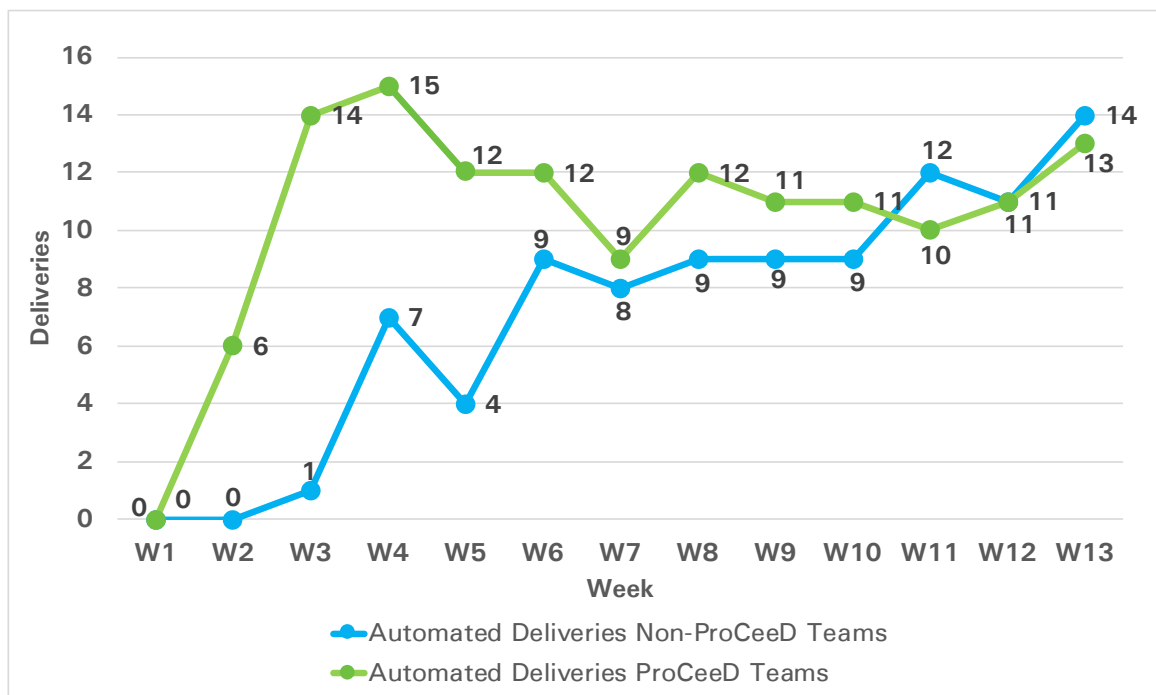


Figure 5.6: Case study I: Amount of prototypes delivered to clients in an automated process.

In the following sections, we inspect the releases carried out by the teams by analyzing the time spans teams took to deliver a release and clients took to provide feedback.

Iteration Length - Delivery Time

We measured the delivery and feedback time intervals for each release delivered by ProCeeD and Non-ProCeeD teams. We only considered the prototypes delivered to the client that received feedback, independently from the channel the client used to provide it (e.g. personal meeting, Prototyper's feedback component or email). For Non-ProCeeD teams we considered a total of 69 releases, for ProCeeD teams we considered 96 releases.

We first compare the time Non-ProCeeD and ProCeeD teams took to deliver revolutionary as well as evolutionary prototypes. Figure 5.7 presents the delivery time for revolutionary prototypes. While the median delivery time of Non-ProCeeD teams was 3.9 days, ProCeeD teams showed a shorter median delivery time of 0.9 days for revolutionary prototypes. In comparison the delivery times for evolutionary prototypes as depicted in Figure 5.8 indicate a more equal median Delivery Time with 1.0 days for Non-ProCeeD teams and 0.8 days for ProCeeD teams. We conclude that while the median delivery time for revolutionary prototypes is lower for ProCeeD teams, the delivery times for evolutionary prototypes differs only marginally between ProCeeD and Non-ProCeeD teams.

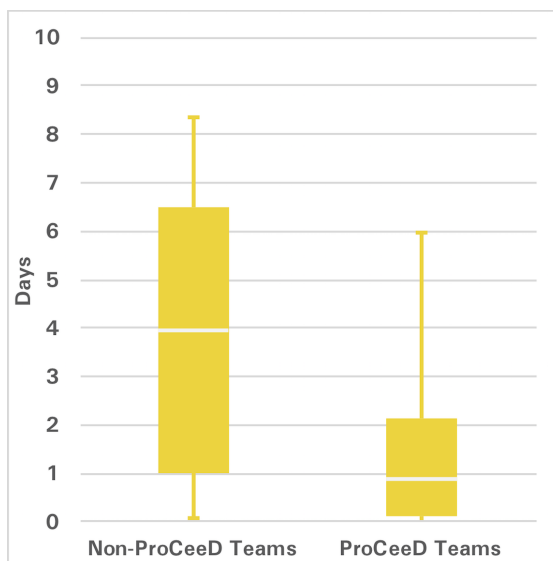


Figure 5.7: Case study I: Delivery times revolutionary prototypes.

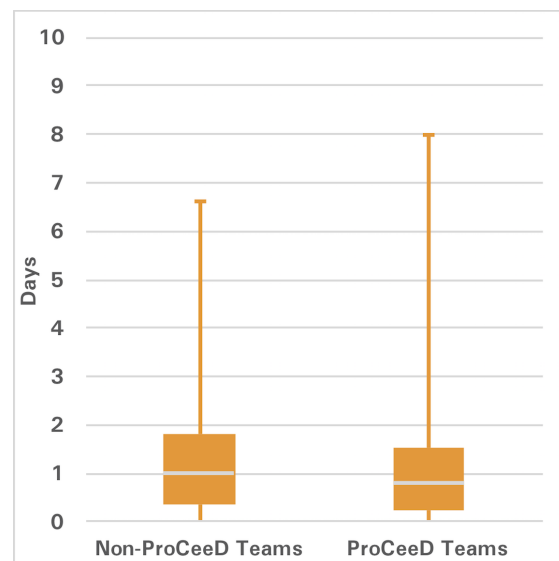


Figure 5.8: Case study I: Delivery times evolutionary prototypes.

Iteration Length - Feedback Time

Next we compare the feedback times of Non-ProCeeD and ProCeeD teams again for revolutionary as well as evolutionary prototypes. Figure 5.9 visualizes the feedback time for revolutionary prototypes. While the median feedback time for Non-ProCeeD teams was 0.2 days, it was 0.8 days for Non-ProCeeD teams. When comparing the feedback time for evolutionary prototypes, as shown in Figure 5.10, the median feedback time for

Non-ProCeeD teams was 0.9 days, and 0.7 days for ProCeeD teams. The results indicate that the feedback time for revolutionary prototypes increased slightly for ProCeeD teams when compared to Non-ProCeeD teams. The feedback time for evolutionary prototypes indicates only a marginal difference.

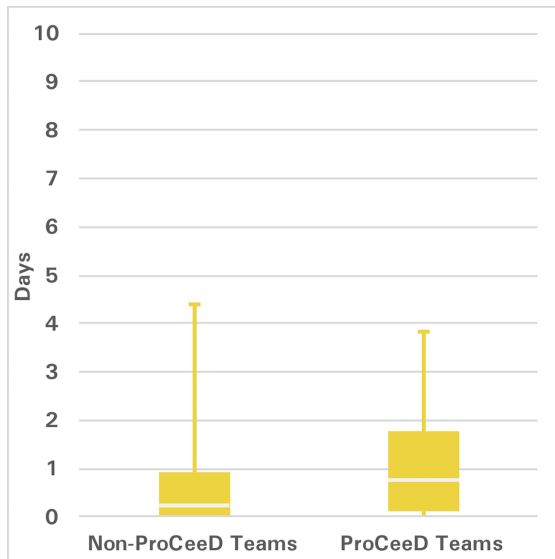


Figure 5.9: Case study I: Feedback times revolutionary prototypes.

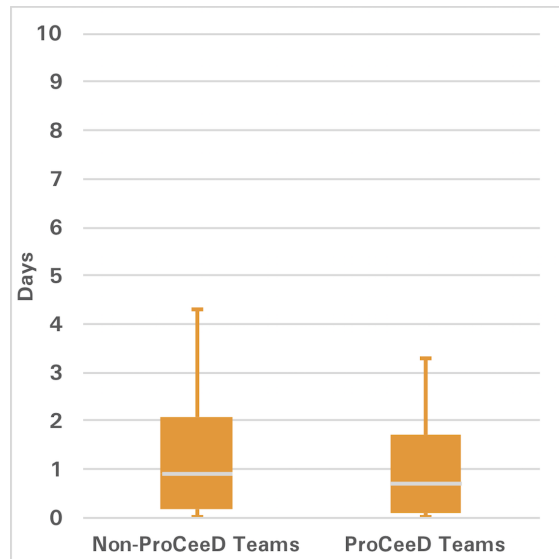


Figure 5.10: Case study I: Feedback times evolutionary prototypes.

Feedback Channels

Finally, we analyzed the feedback channels chosen by the clients. Figure 5.11 depicts the results. All teams used personal meetings to gather feedback from the client roughly every two weeks. Prototyper’s in-app feedback component, email and the Slack instant messaging service were used by half of the teams. Feedback via the wiki system was provided only by the client of one team, the issue tracker was not used to provide feedback.

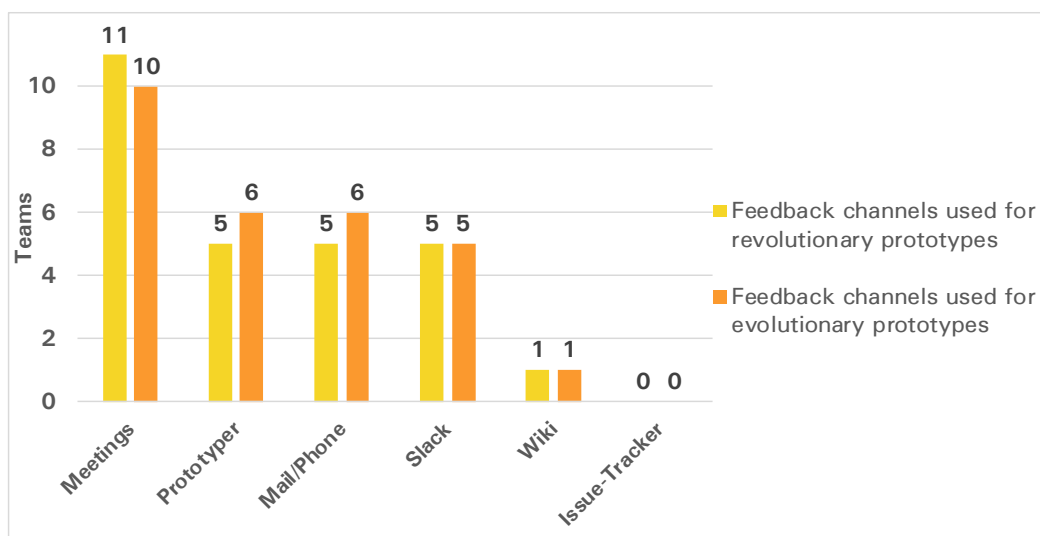


Figure 5.11: Case study I: Feedback channels.

5.2.3 Qualitative Results

In this section we present the results of the questionnaire sent to the ProCeeD teams to complement the quantitative evaluation presented in Section 5.2.2.

Tasks

We first analyzed how often each participant performed a certain tasks in Prototyper. Participants answered three questions about whether and how often they used features of Prototyper in their personal workflow. Figure 5.12 visualizes the responses. The first question dealt with the issue whether the participants downloaded a release on their own device. 51% of the participants downloaded a release 7 or more times, 20% 3-6 times. 29% downloaded a release only 1-2 times or not at all. The second question asked if the participant used Prototyper's in-app feedback component. 19% of the participants provided feedback using Prototyper 7 or more times, 17% 3-6 times. 64% used the feedback function only 1-2 times or not at all. Question three asked the participants whether they used Prototyper to lookup feedback provided by the client. 46% of the participants looked up client feedback for a release 7 or more times, 21% 3-6 times. 33% looked at client feedback only 1-2 times or not at all.

We conclude that while most of the participants downloaded releases and reviewed client feedback multiple times, Prototyper's in-app feedback function was hardly used by the participating students.

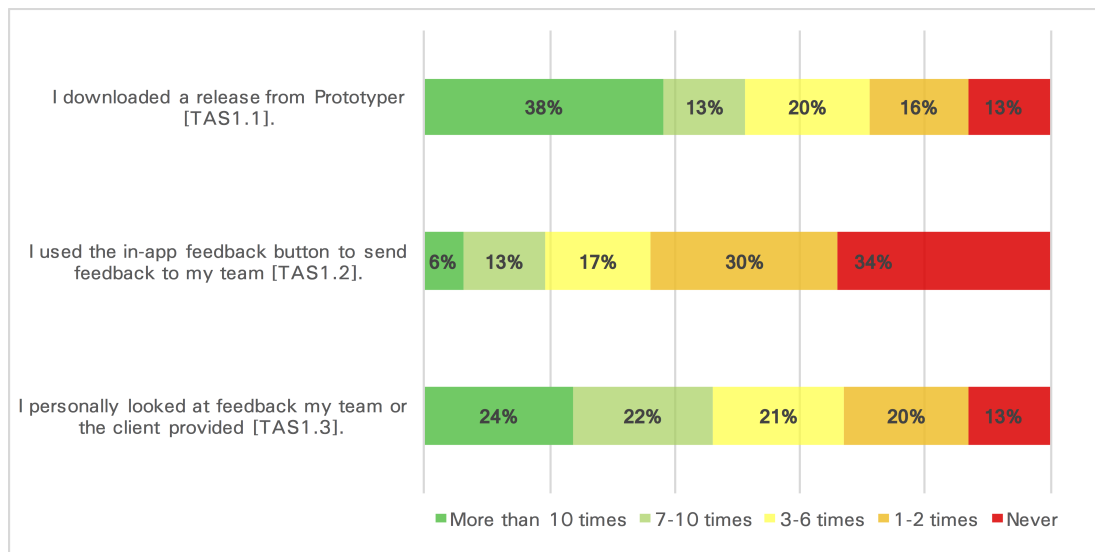


Figure 5.12: Case study I: Prototyper usage (single choice).

Delivery Automation Workflow

To assess ProCeed's delivery automation workflow we inquired about the participants' opinion on five statements depicted in Figure 5.13 and 5.14. 85% of the participants strongly agreed or agreed that Prototyper allowed them to use the same delivery workflow from the first wireframe up to the delivery at the end of the project, 4% had a neutral opinion and 11% disagreed with the statement. 69% of the participants strongly agreed or agreed that Prototyper allowed them to deliver revolutionary prototypes faster compared to other tools they are aware of. 5% had a neutral opinion and 26% disagreed with the statement. 67% of the participants strongly agreed or agreed that Prototyper made their team's delivery workflow faster compared to other tools they are aware of. 7% had a neutral opinion, 26% disagreed or strongly disagreed with the statement.

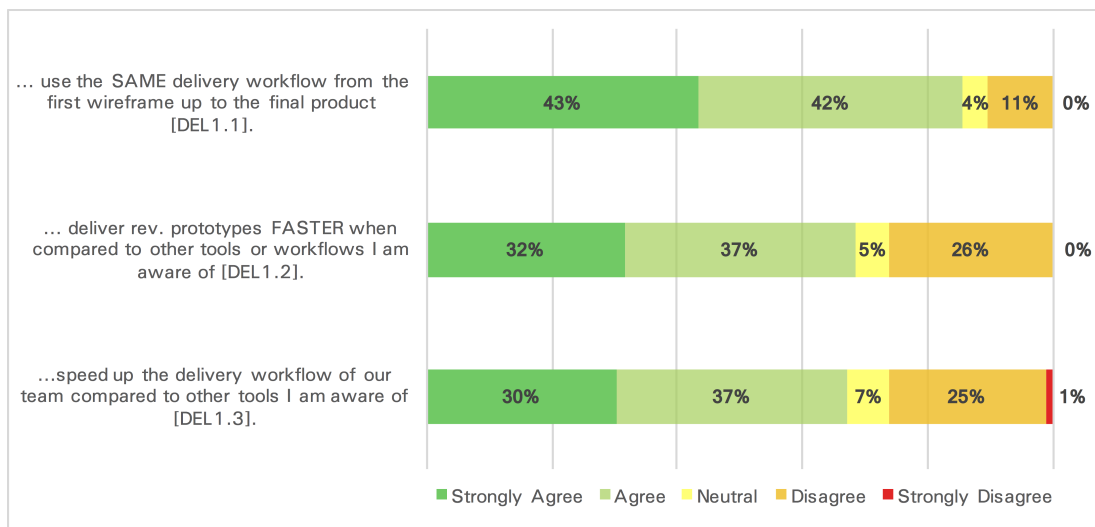


Figure 5.13: Case study I: Statements delivery automation 1.1 - 1.3 (5-likert).

In addition we asked if participants knew of the Prototyper's concept of managing multiple user groups. 87% of the participants strongly agreed or agreed to the statement that Prototyper allowed them to manage different user groups. 12% had a neutral opinion and 1% disagreed. When asked if they always knew when a release was delivered to a user group 75% agreed or strongly agreed, 22% had a neutral opinion and 3% disagreed.

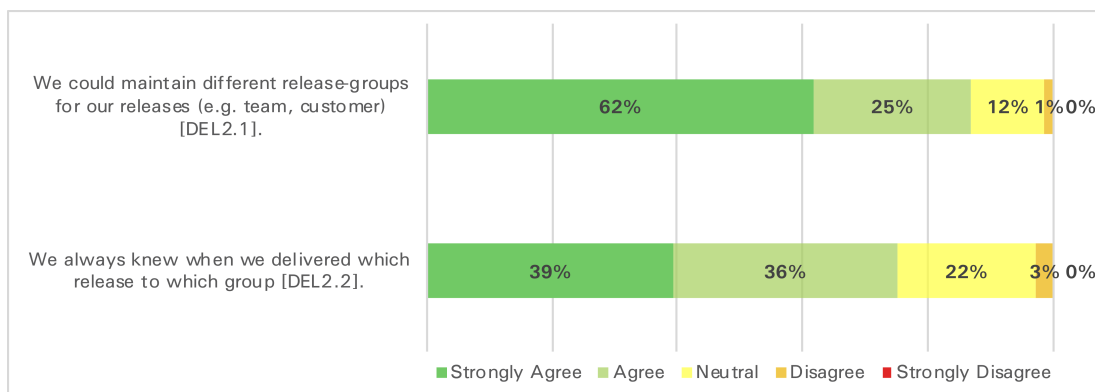


Figure 5.14: Case study I: Statements delivery automation 2.1 - 2.2 (5-likert).

Requirements Exploration

To assess ProCeeD's requirements exploration workflow we asked the participants to state their opinion on the six statements depicted in Figure 5.15 and 5.16. 65% of the participants strongly agreed or agreed that they see an overall use of Prototyper for exploring requirements. 27% had a neutral opinion. 8% disagreed or strongly disagreed with the statement. Second we asked participants to state their opinion on two statements regarding Prototyper's functionality. 75% of the participants strongly agreed or agreed on the statement that Prototyper allows them to deliver multiple releases to a user-group at the same time, 4% had a neutral opinion, 21% percent disagreed or strongly disagreed. Moreover, 52% of the participants agreed that the delivery of multiple releases provided a benefit for collecting feedback from the client. 26% had a neutral opinion. 22% of the participants disagreed or strongly disagreed with the statement.

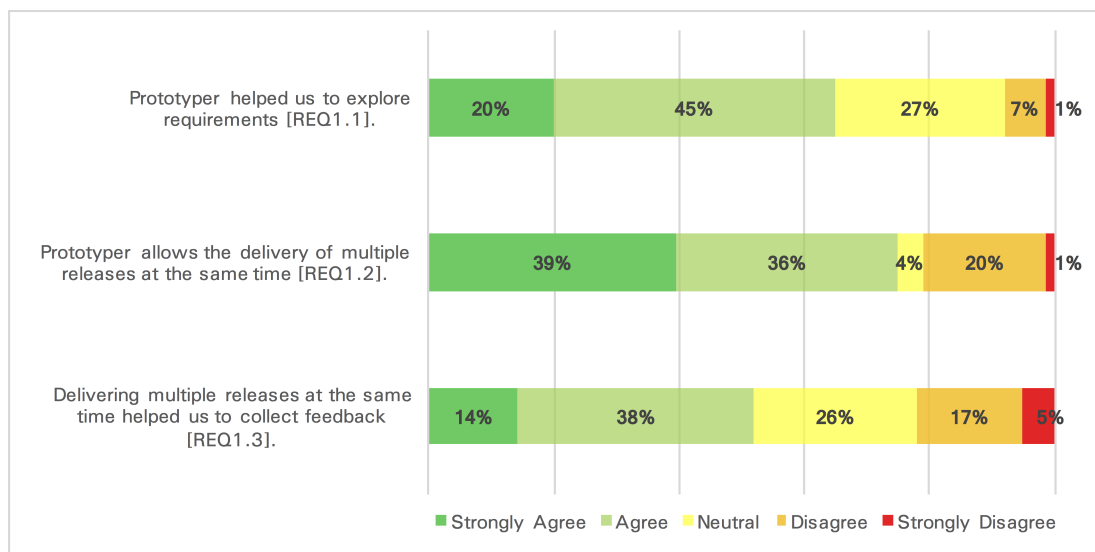


Figure 5.15: Case study I: Statements requirements exploration 1.1 - 1.3 (5-likert).

We then asked if their client knew from where to get the a releases delivered by the team already in the first week of the course. 51% of the participants strongly agreed or agreed on the statement, 29% had a neutral opinion, 20% percent disagreed or strongly disagreed. In the second question 89 % of the participants strongly agreed or agreed that the team applied build promotion by evaluating each release internally first before delivering it to the client. 5% had a neutral opinion, 6% percent disagreed or strongly disagreed. In the third question we asked wether the participant know how to relate a client feedback item to a release. 59 % of the participants strongly agreed or agreed, 34% had a neutral opinion, 7% percent disagreed or strongly disagreed.

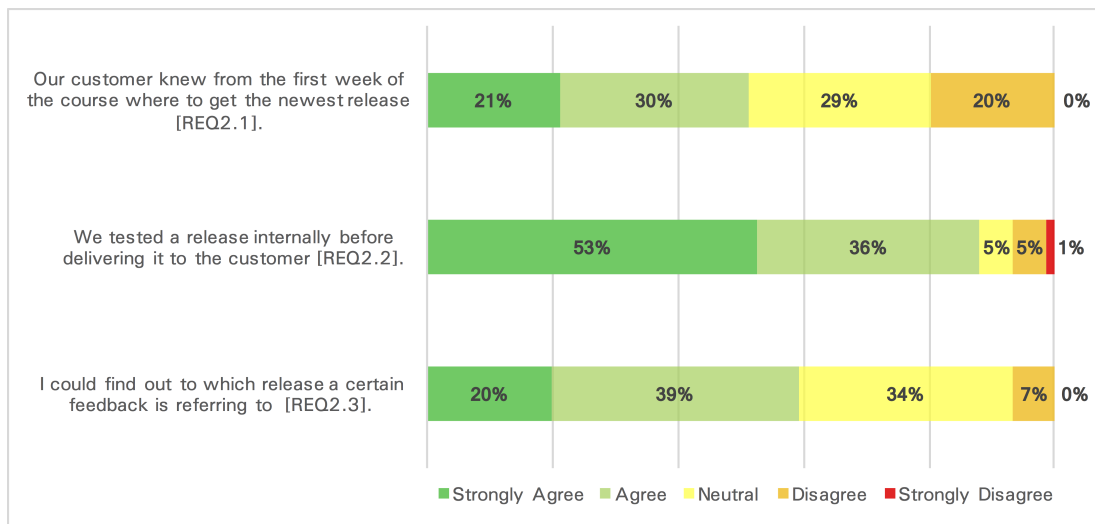


Figure 5.16: Case study I: Statements requirements exploration 2.1 - 2.3 (5-likert).

Integrations

Finally we presented the survey participants with six statements regarding Prototyper's integration with development tools and asked them if they found these integrations useful for their development workflow. The answers are depicted in Figure 5.17 and 5.18.

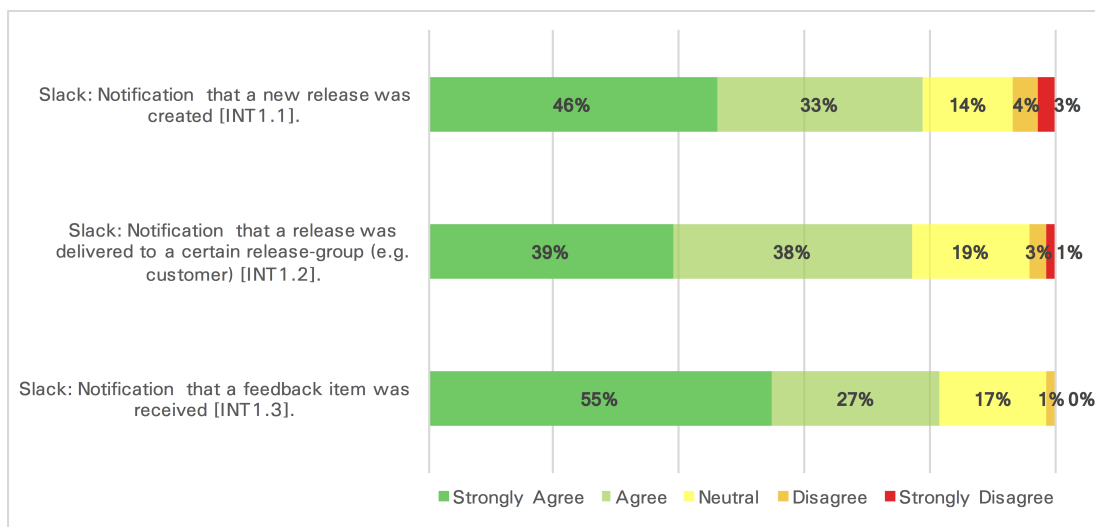


Figure 5.17: Case study I: Statements integration with developer tools 1.1 - 1.3 (5-likert).

First we presented the participants with three statements about Prototyper's integration with Slack³, the main communication tool used during the capstone course. Most of the participants strongly agreed or agreed that the notifications Prototyper provided into the Slack channels of the teams about a release being created (79%), delivered (77%) or a feedback to a releases being received (82%) were useful to them. The remaining participants had mostly a neutral opinion, less than 8% disagreed.

³<http://www.slack.com>

We then asked the participants about the integration with Atlassian Bamboo⁴, the integration server used during the course. 55% of the participants strongly agreed or agreed that the traceability from a release created in Prototyper to the corresponding build and commit was useful to them. 38% had a neutral opinion, 7% disagreed.

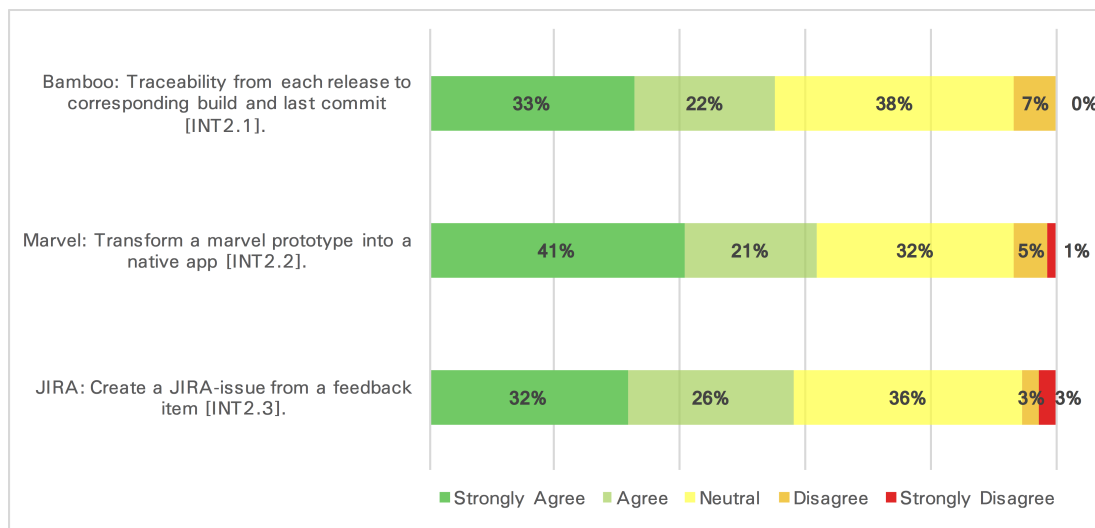


Figure 5.18: Case study I: Statements integration with developer tools 2.1 - 2.3 (5-likert).

Regarding Prototyper's capability to transform a prototype created with Marvel App⁵, a prototyping tool used during the course, 62% of the participants agreed or strongly agreed that it was useful for them. 32% had a neutral opinion, 6% disagreed or strongly disagreed. Finally, we asked about Prototyper's functionality to copy user feedback from Prototyper to Atlassian JIRA⁶, the issue-tracker used during the course. 58% of the participants strongly agreed or agreed that this feature was useful for their development workflow, 36% had a neutral opinion. 6% disagreed or strongly disagreed with the statement.

5.2.4 Discussion

The quantitative results presented in Figure 5.5 indicate that ProCeeD teams delivered an increased amount of revolutionary prototypes. Furthermore ProCeeD teams were able to deliver revolutionary prototypes using an automated process. Due to the fact that also Non-ProCeeD teams delivered evolutionary prototypes using an automated and repeatable process, the amount of evolutionary prototypes delivered did not change. The results depicted in Figure 5.6 support this finding. It shows that ProCeeD teams delivered more revolutionary prototypes than Non-ProCeeD teams in the first six weeks of the capstone course: most of the user interaction design is carried out by the teams during this time. Therefore, revolutionary prototypes are frequently used.

In later project stages and when teams mostly delivered evolutionary prototypes, ProCeeD and Non-ProCeeD teams did not show a notable difference in the amount of

⁴<http://www.atlassian.com/bamboo>

⁵<http://www.marvelapp.com>

⁶<https://www.atlassian.com/jira>

prototypes delivered. The survey results indicate that the majority of the participants agreed that Prototyper allows the delivery of revolutionary and evolutionary prototypes using the same workflow. Therefore, we gained support for hypothesis H1 which states that using ProCeeD, developers can adopt a common delivery process for revolutionary as well as evolutionary prototypes.

The quantitative data also indicates that ProCeeD teams had a shorter delivery time for revolutionary prototypes. Revolutionary prototypes were delivered in less than one day after they were finished compared to Non-ProCeeD teams which took four days. We observed that the Non-ProCeeD teams presented most of the revolutionary prototypes in personal meetings with their client and did not deliver them upfront: they finished the prototype some days before the next meeting but delivered them only right before or even during the meeting. ProCeeD teams delivered revolutionary prototypes more quickly after they finished them. We could assume that ProCeeD teams did not evaluate a prototype properly internally before delivering it but the questionnaire responses indicate that ProCeeD teams mostly delivered a prototype to their team first and promoted it to their client afterwards.

In summary, ProCeeD teams show a decrease of the prototype delivery time (see Figure 5.4) for revolutionary prototypes. The results support hypothesis H2 which states that ProCeeD lowers the time developers need to deliver prototypes and allows shorter prototyping iterations.

During our case study ProCeeD teams showed a two-fold increase over Non-ProCeeD teams in deliveries of revolutionary prototypes to their client. At the same time, they received feedback from the clients on more releases, 69 for the Non-ProCeeD teams and 96 releases with feedback for the ProCeeD teams. The measured feedback times indicate that ProCeeD teams took longer than Non-ProCeeD teams to gather feedback on a revolutionary prototype. The feedback times for evolutionary prototypes did not differ. Non-ProCeeD teams delivered revolutionary prototypes mostly in personal meetings and they received feedback on them right away. We conclude that ProCeeD teams received feedback on more releases than Non-ProCeeD teams. As we did not evaluate the quality of the received feedback we can only partially support hypothesis H3 which states that ProCeeD allows developers to collect more and better feedback.

Finally, we evaluated how Prototyper and ProCeeD are adopted by developers. Almost every participant used Prototyper to download new releases. 78% of the survey participants own a device matching the target environment of the capstone course, consequently only these students could actually download a release. The majority of participants agreed that Prototyper made their development workflow faster, in particular when delivering revolutionary prototypes. Given the fact that prototyper's feedback component was only one of many feedback channels used by the teams, the Prototyper's in-app feedback

component (see Section 4.1) was only rarely used by the developers. We observed that developers mainly used communication channels such as instant messaging or personal meetings to provide feedback.

Concerning Prototyper's feedback management component we conclude that two thirds of the participants actively used Prototyper's functionality to receive feedback from the clients. The questionnaire results further indicate that the majority of the participants understood and applied the concept of multiple release groups. Regarding Prototyper's requirements exploration workflow, participants' opinions varied concerning whether the delivery of multiple releases at the same time helped them to gather feedback. While the requirements exploration using multiple releases showed mixed results, we can still provide support for hypothesis H4 that developers see a benefit of adoption ProCeeD in their workflow.

5.3 Case Study II: Process Metrics

In this case study we describe how we applied ProCeed’s process metrics workflow to improve the manageability of a multi-project capstone course in a university context. Section 5.3.1 describes the study design. Section 5.3.2 presents the results of applying ProCeed’s process metrics in a capstone course with more than 100 students developing applications in 10-12 parallel projects over the course of three months. Section 5.3.3 discusses our experiences on how we selected and used process metrics during the course. This case study has been published in [ADB16].

5.3.1 Design

In this case study we evaluate how ProCeed can help course instructors to manage project-based software engineering courses which involve workflows like Continuous Delivery and Continuous Prototyping. Teaching and monitoring the adoption of such workflows can be time-consuming, which is why the instructor needs a quick way to assess each project’s status and identify potential problems.

In the following section we first summarize two workflows, namely Continuous Integration and Continuous Prototyping and second explain how we used ProCeed to select and compute metrics to measure the adoption and usage of each workflow in the project teams. For the selected the metrics, we analyzed each of the workflows with regard to its desired outputs based on the techniques suggested by [Kan+04; Gra94a; FN00].

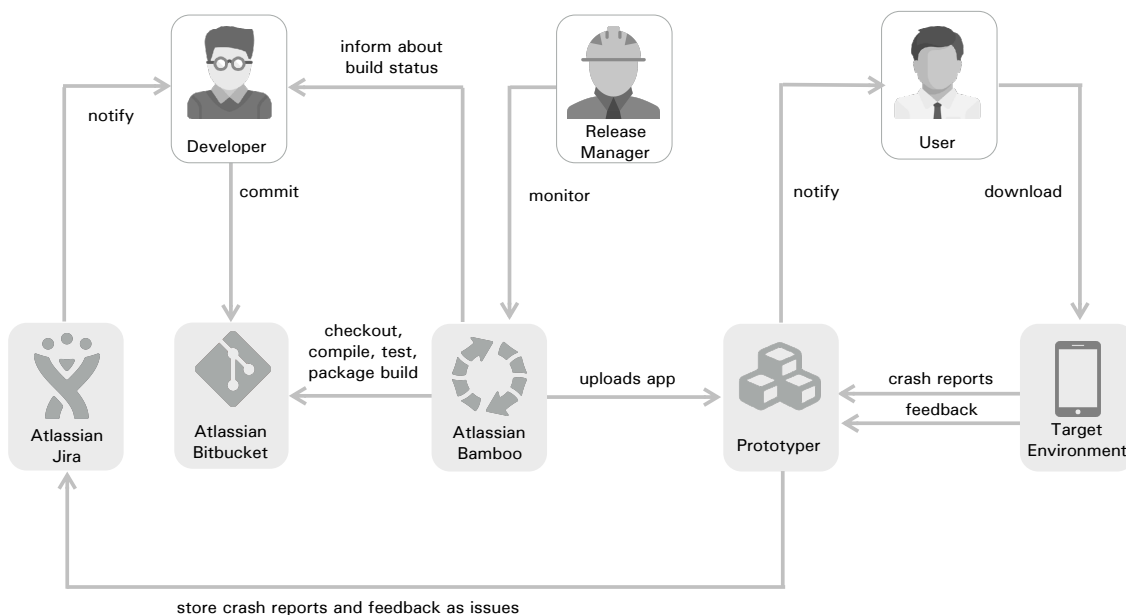


Figure 5.19: Case study II: Development workflow (adapted from [BKA15])

Continuous Integration

To introduce Continuous Prototyping a team needs to adopt Continuous Integration practices first. As part of this workflow we used in the course a dedicated system which executes Unit-Tests as well as automated UI-Tests after each commit in a team's repository. If an error occurs, the author of the commit is notified, so that he can fix the code and the according build as soon as possible. The workflow is described in detail in [BKA15]. Experiences from past courses show that the reasons for a failing build can be manifold, ranging from coding errors over configuration errors of the build server. As Continuous Integration focuses on the key principle of getting the build to pass successfully again as soon as possible, we measure this workflow with the following two metrics.

Metric: Continuous Integration - Time to fix

We calculate the time to fix, which is the average period of time between a failed build and the first successful build on the team's main development branch [DMG07]. A high time to fix could e.g. indicate that a team does not check the status of the integration server, or that the developers are not experienced enough to fix errors in the build pipeline. If the time to fix increased significantly over the past weeks, for instance, the instructor can take a look at the corresponding project and talk to the team to find out more about the root cause.

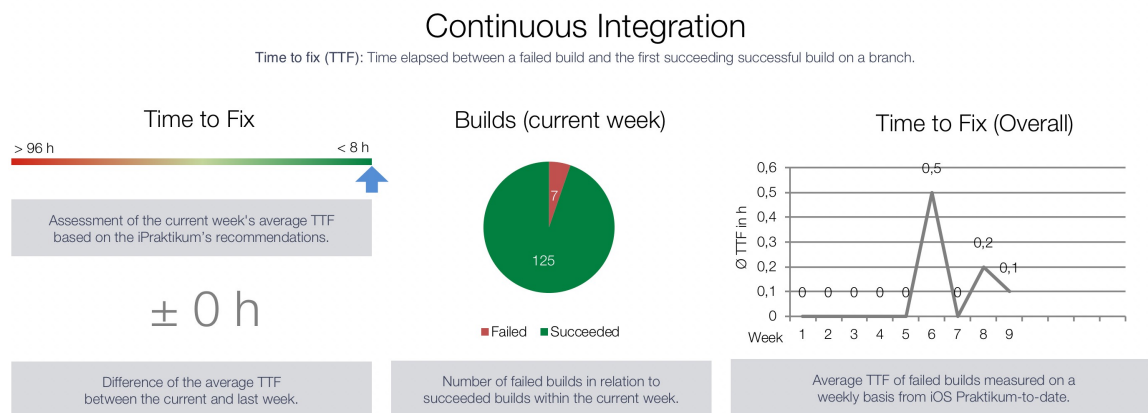


Figure 5.20: Case study II: Metrics Continuous Integration.

Metric: Continuous Integration - Builds

By showing the absolute numbers of successful and failed builds we allow the course instructor to analyze a team's Continuous Integration process. A large amount of failed builds could e.g. indicate that the team was unable to interpret the errors provided by the integration server or that the server itself did not work as expected.

Continuous Prototyping

To evaluate the adoption of ProCeeD and Continuous Prototyping we evaluated the deliveries of a project team to their clients. We encourage teams to deliver prototypes to their client regularly in order to define and refine their requirements, e.g. when they implemented a new feature. Figure 5.21 shows the metrics we calculated using ProCeeD's Process Metrics Workflow:

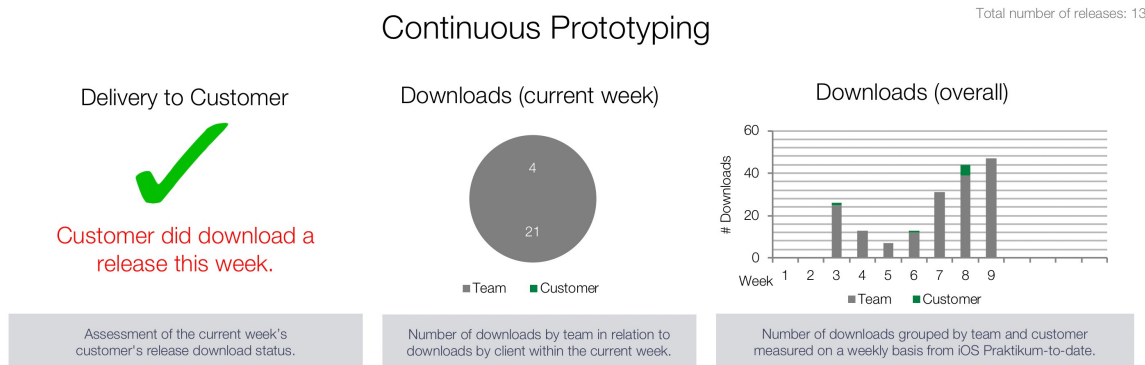


Figure 5.21: Case study II: Metrics Continuous Prototyping.

Metric: Continuous Prototyping - Delivery to Client

With this indicator, we determine each week if a client has downloaded a release of the team's application at least once. A successful download by the client can be an indicator for the grade of their current involvement. If a team does not have any download by their client for weeks, an instructor can inquire about possible root causes. These can range from a problem in communication with the client to delivery problems due to the nature of the project. For instance, if the team develops a complex system with various sensors or other hardware devices involved, a delivery to the client can be challenging.

Metric: Continuous Prototyping - Downloads

To evaluate a build of the system on a mobile device before delivering it to the client, the teams used Prototyper to download the build of their application to their devices before delivering it to the client. This allowed them to minimize the amount of errors that can happen during the delivery of a new product increment [HF10]. In addition we measure the number of downloads to team members' devices to determine if they make use of the promotion workflow.

5.3.2 Results

During the course of one semester we combined the collected metrics on a single sheet for each project so that a team can evaluate their development workflow as a whole. We believe that an indicator considered in isolation can not accurately predict project success [Kan+04]. We generated the reporting sheet for each project team on a weekly basis during the length of the course and used it to identify projects where certain workflows were not adopted as expected. After taking a more detailed look to identify possible root causes, we talked to the project leader or coach of the team to find a solution. Multiple times we found that even though some indicators pointed towards a problem, the team had very good reasons to work the way they did. In these cases we stuck to one of the key principles of the agile manifesto: "Individuals and interactions over processes and tools" [Bec+01], and did not ask the teams to change their approach.

In personal interviews the instructors and project leaders stated that the metrics sheet gave them a quick overview over the project status. If they noticed an irregularity, e.g. a high number of failed builds, they could go into the respective systems and investigate the root causes or talk to the responsible members of the development team. Therefore, the metrics were often a good first indicator and proved to be a tool to oversee several projects at once.

5.3.3 Discussion

As a project-based organization our capstone course is run by instructors and students in multiple roles. We think that choosing the right target audience for process metrics upfront is an important success factor. Instructors and project leaders are more experienced than team coaches and developers which are students and therefore much usually less knowledgeable in the field of project management. We think that students who have not yet experienced how a project's characteristics can influence such indicators would tend to interpret the metrics too literally, while instructors have the proper mindset to use the metrics as a basis for making decisions. For instance, making the number of commits of each team member available to the participants might drive them to increase their commit count by e.g. making unnecessary commits, because many of them fail to understand the limited explanatory power of this metric when considered in isolation.

In addition to limiting the use of metrics for direct comparisons *within* teams, we do not think that comparing whole teams directly *to each other* is useful to the outcome of the course. Each team works on a different project with its particular challenges, and they can take very different paths to reach their goal [DKA14]. Our experience shows that a team's characteristics ranging from the type of problem to solve to the personality of the team members can be quite different, and they can and should not be ranked directly against each other using performance indicators. In summary, we recommend to apply the following rules to adopting process metrics in software engineering project courses: metrics should be used and exposed only to instructors and should not be accessible by students. A high-level overview over the individual projects allows the instructors to

recognize and react to problems in the adoption workflows version control, continuous integration and continuous delivery early on. Nevertheless, we think that regardless of how thoroughly the data is collected, a simple number should never be the only indicator of project progress when it comes to teaching software engineering in a real-world setting. We therefore conclude that these process metrics should be applied with caution, especially when it comes to grading and assessing students' performance. Instructors should treat the metric as a first indicator of a possible problem and always investigate the root causes by e.g. speaking to the team members.

In the case study we showed a possibility how instructors can use process metrics created using ProCeeD to reduce the amount of time they need to monitor the adoption of Continuous Integration and Continuous Prototyping in capstone courses. In personal interviews with the instructors of a capstone course we found anecdotal evidence that ProCeeD's process metrics workflow helps them to manage the delivery workflow of multiple projects at once by checking a standardized metrics sheet on a weekly basis. We therefore gained support for hypothesis H5a which states that ProCeeD improves the traceability of the software delivery process and therefore reduces the time needed to monitor multiple projects at the same time.

5.4 Case Study III: Industry

In this case study we applied Prototyper in a commercial project with a partner from the pharmaceutical industry.

5.4.1 Design

We introduced Prototyper and ProCeeD in a project concerned with the development of a mobile diabetes management application *DiaApp*. At the beginning of the evaluation period, *DiaApp* 1 had been launched for two months and the team was getting ready to begin the design and development of version 2. The development team consisted of four persons including an interaction designer. On the client side three people were involved. The client was located remotely and only met with the development team in person once per month. The team also involved a group of testers consisting of end users as well as colleagues of the clients in the project. The involved end users were real patients with diabetes and evaluated the new dashboard from the perspective of a person with diabetes. The development of version 2 was carried out following Scrum as an agile process model, and the project ran over a period of four months. As target environment of the application the Apple iOS platform was used. The scope of the project included the rework of the interaction design of a central component of *DiaApp*, the *Dashboard*. With the help of the dashboard the app user can visualize health parameters like measured blood sugar levels, injected insulin doses and carbohydrates eaten. In the following section we describe our observations during the study regarding the development of the next version of the dashboard component of *DiaApp*.

5.4.2 Results

The team began the development with a discussion about user feedback regarding the already launched version of *DiaApp*. They then structured the requirements of the clients in a product backlog. During the development of version 2 the team created revolutionary as well as evolutionary prototypes for the new dashboard. Every time they wanted to gather feedback from their colleagues, clients or a group of end users, the team sent out a release using Prototyper. Table 5.3 shows the total amount of releases delivered to developers, clients and users.

Table 5.3: Case study III: Releases delivered for *DiaApp* 2

	Rel. to Developers	Promotions to Clients	Promotions to End Users
Rev. Prototypes delivered	17	11	3
Ev. Prototypes delivered	77	14	4
Feedback Items received	131	95	12

As the team applied ProCeed's delivery automation workflow, they delivered each release first internally and promoted it some time later to the clients or end user. The team promoted 11 revolutionary and 14 evolutionary prototypes to the clients. 3 revolutionary prototypes and 4 evolutionary prototypes were even delivered to end users. The team received a total of 221 feedback items collected using Prototyper's feedback collector component or via other feedback channels like email. 131 items were written by developers, 95 by clients and 12 by end users. 65% of the feedback items contained annotated screenshots.

In the following we describe our observations regarding the development workflow applied by the team. Figure 5.22 shows a simplified release plan visualizing how the team structured their release workflow. The development began with all user groups using R1, which has been released as DiaApp 1 earlier on. After creating three iterations of a revolutionary prototype, the developers decided to deliver RV4 to the client. To allow the client to compare the EV1 and RV4, the developers decided to deliver Release R2 in addition to the already delivered R1. The team received feedback by the client and incorporated it into the prototypes RV5 and RV6. After receiving only positive feedback for R3 from the clients, the team decided to send the release to the end users who were actual patients. This process continued until all requirements were met and implemented. EV8 was released as R4, tested and launched as DiaApp 2.

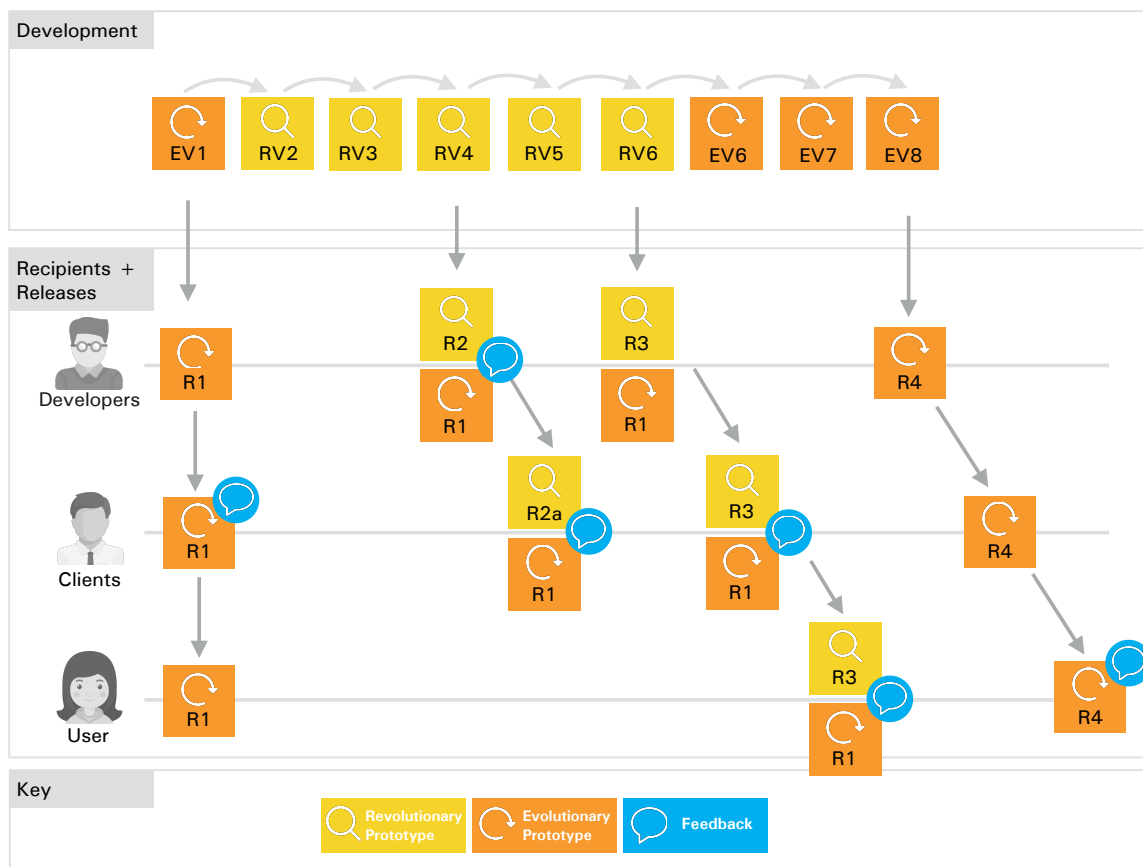


Figure 5.22: Case study III: Release plan for DiaApp - example.

Figure 5.23 presents screenshots of four iterations EV1, RV4, RV6 and EV8 which were delivered to the clients or end users.

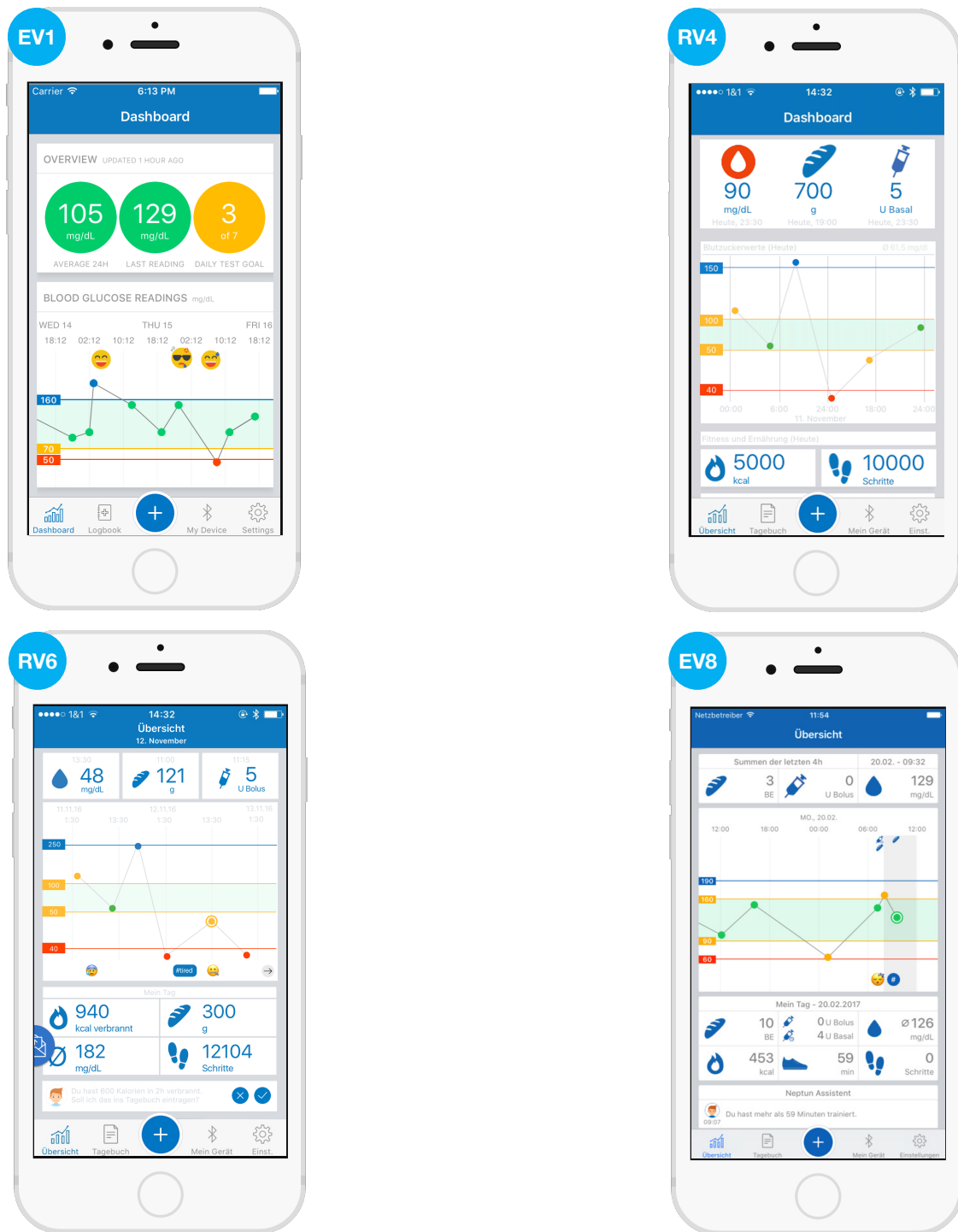


Figure 5.23: Case study III: Four prototyping iterations of the DiaApp dashboard.

5.4.3 Discussion

In personal interviews, the three clients and the developers agreed that using Prototyper improved the prototyping process compared to other approaches they were aware of as follows: clients were able to receive and download revolutionary and evolutionary prototypes using the same workflow. They particularly liked the way to provide feedback in-app because this simplified the process to e.g. comment on user interface design elements. One client stated that he liked the approach of having executable, revolutionary prototypes as mobile apps because he could quickly install them on his colleagues' devices. Another mentioned that evaluating prototypes by installing an actual application on his smartphone became part of his weekly routine.

Developers stated that delivering each prototype using the same workflow especially speeded up the development and helped them to organize user feedback at a single place. We observed that end users, who evaluated the revolutionary prototypes of DiaApp 2, gave different feedback than clients or developers: while clients and developers were already focused on the implementation of the requirements they agreed on, end users also provided feedback which was rather focused on new features or improvements to their daily workflow.

In this case study, we showed how ProCeeD and Prototyper can be used to deliver revolutionary as well as evolutionary prototypes to developers, clients and actual end users using a common delivery workflow. In personal interviews with the clients and developers we gained support for hypothesis H1 which states that ProCeeD allows a common delivery process for revolutionary and evolutionary prototypes.

In addition, we showed how the project team applied the same feedback mechanism for revolutionary and evolutionary prototypes as well as for each of the user groups. Overall a total amount of 221 feedback items were collected. Hypothesis H3 can therefore be further supported. It should be noted that as we only included a few end users in the case study, the implications of exposing early, revolutionary prototypes to hundreds of end users remains future work.

5.5 Case Study IV: Storyboard-based Requirements Elicitation

In this case study we applied the ProCeed workflows for a storyboard-based requirements elicitation approach in a two-week software engineering course [ASF17]. Storyboards are a well-established practice in the filming industry to organize and sketch out movies. When used for software development, storyboards allow developers to visually structure a project's requirements. Each frame of the storyboard describes a requirement and the subsystems involved in the implementation of the requirement.

5.5.1 Design

The course was conducted with 15 students and 4 instructors. The task of the students was to create a series of prototypes of a human-centric cyber-physical system which supports humans in hazardous environments [SSBV16]. We provided the students with equipment such as sensor kits, soldering machines, a lab room and a 3D printer. Furthermore, we introduced them to the ProCeed workflows. The course was conducted using an agile methodology based on Scrum. A visual storyboard served as a product backlog to structure the development and delivery. Every morning the current state of the storyboard was shown to the students during the stand-up meeting. During this meeting, the team prioritized what to do next and which frames of the storyboard should be implemented. As soon as a storyboard frame was finished, the students acted out the frame in a small theater play which was filmed. This frame of the storyboard was then replaced by the filmed clip.

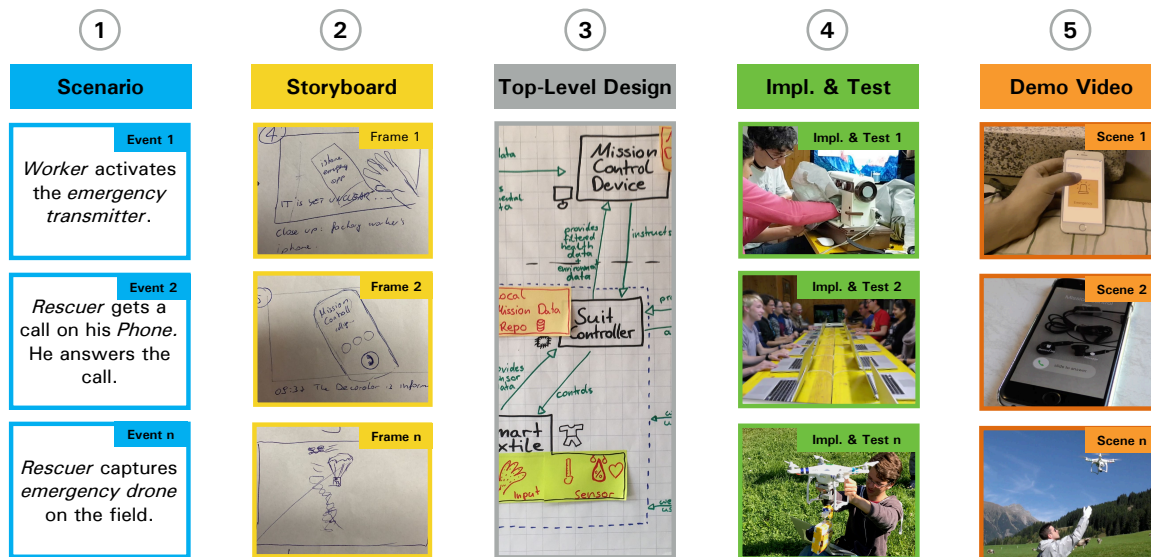


Figure 5.24: Case study IV: Visual backlog creation process (adapted from [ASF17]).

5.5.2 Results

At the beginning of the course, we asked the students to define a visionary scenario for the system to be developed [RC09], [BKA15]. In the scenario, the students described the event flow to be carried out by the actors of the proposed system (see Figure 5.24.1). We asked the students to also propose and describe features even if they were considered not realizable during the two weeks of the course. Based on the event flow the students created the initial storyboard shown in Figure 5.24.2. Each frame shows a specific drawing which represents the shots planned for the movie. The storyboard frames also served as a visual product backlog. An initial architecture model was created in parallel to the visual product backlog and kept in sync in subsequent iterations (see Figure 5.24.3). The architecture model represents an informal communication diagram where each subsystem is labeled and depicted with an icon. Arrows between the subsystems are used to indicate message exchange. During each iteration, the architecture model and the storyboard prototype were refined. Each change in the storyboard prototype may influence the architecture model and vice versa. Figure 5.25 shows an iteration of the architecture model.

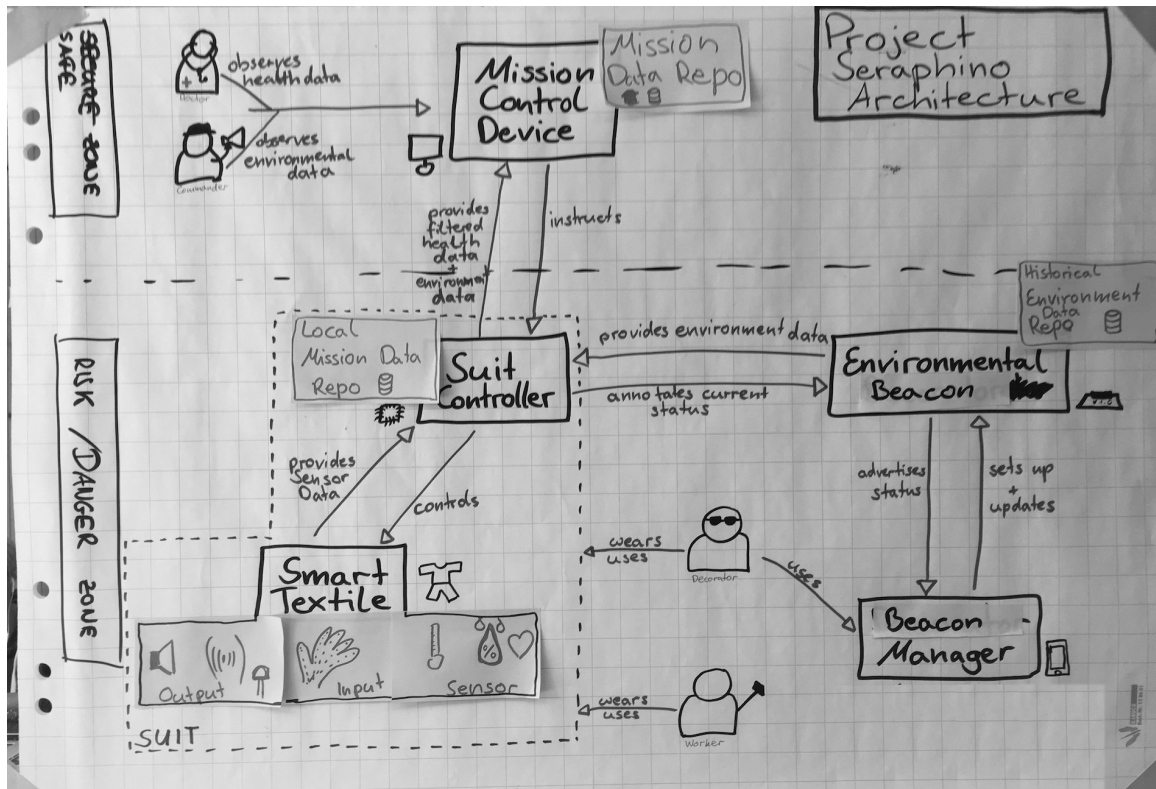


Figure 5.25: Case study IV: Top-level architecture (example).

For the implementation of the executable prototype, the students chose a frame from the storyboard and the according subsystems from the architecture model. Afterwards the executable prototype was delivered using Prototyper to actual devices (see Figure 5.24.4). In addition, an integration test was conducted to check if the subsystems involved in one frame were ready to be filmed. After the unit and integration tests had passed,

the scene was staged, filmed, and the hand-drawn frame in the storyboard was replaced with the filmed clip (see Figure 5.24.5). The current iteration of the animated storyboard was then delivered to the instructors as a movie. Hand-drawn prototypes were digitized and, like evolutionary prototypes implemented in code, delivered to the course instructors via Prototyper. After several iterations, more and more drawn frames were replaced with the actual movie clips, showing the implemented system. This process was repeated until all drawn frames in the storyboard were replaced by filmed frames. Figure 5.26 shows a snapshot of the storyboard prototype after five iterations. Parts of the frames are still hand-drawn and not implemented or filmed yet, others are already filmed and the functionality of the executable prototype shown in the frame is already implemented.

5.5.3 Discussion

In this case study we presented an approach to use ProCeeD workflows for the development and delivery of prototypes based on storyboards in terms of demo movies. Each demo movie allowed the students to integrate the component of the system as soon as their implementation was finished, while also visualizing the current progress. Using Prototyper students were able to quickly deliver even multiple revolutionary and evolutionary prototypes and use them to create the demo movie. Our results further support hypothesis H1 as we applied ProCeeD to automate the delivery of revolutionary prototypes created with a storyboarding approach. In addition, we found that visual product backlogs facilitated a common understanding of the problem to be solved in even in an early stage of the project while still allowing the development of requirements iteratively and incrementally. We believe that the use of storyboards is a promising approach for agile software development, in particular if interdisciplinary teams are involved.



Figure 5.26: Case study IV: Visual backlog showing the system after five iterations.

5.6 Summary

Table 5.4 summarizes the results of the four case studies and presents the evaluation results of each hypothesis we stated in Section 5.1.1. In Case Studies I, III and IV we presented quantitative as well as qualitative evidence that ProCeeD allows developers to deliver revolutionary as well as evolutionary prototypes using a common delivery process (H1). We further showed quantitative evidence that ProCeeD allows developers to reduce the length of a prototyping iteration (H2) in Case Study I. We can only partially support hypothesis H3 in Case Study I and III: while teams who adopted ProCeeD showed a shorter overall iteration length when compared to Non-ProCeeD teams we could not show an improvement of the feedback quality. While we showed quantitative evidence that teams adopting ProCeeD receive more user feedback than teams not using ProCeeD, we did not analyze the quality of the received feedback. With Case Study I we showed evidence that developers see a benefit of applying ProCeeD in their development workflow (H4). We finally evaluated ProCeeD's process metrics workflow in Case Study II and showed that it improves the traceability of the software delivery process in multi-project organizations (H5).

5.7 Threats to Validity

We see the following limitations regarding the results of the four case studies presented. Case Study I was conducted in a large university capstone course. With regard to the quantitative data and the underlying usage of ProCeeD we saw the following threats to validity: While we selected students with various prior experience and from different fields of study a selection bias regarding the participating students cannot be ruled out. When we computed the delivery and feedback time spans, our starting point was the creation of a prototype. Hence, our results do not consider a possible overhead of creating the revolutionary prototypes. The results of the qualitative study part are based on subjective ratings by the participants. While we conducted the study after the grading was finalized, we cannot rule out a bias in the answer of the students. In Case Study II we presented how process metrics allow the instructors to maintain an overview over the projects and intervene when problems arise. We presented only anecdotal evidence that the metrics described increase the manageability of such project courses. The study described the application of ProCeeD's process metrics workflow from the perspective of the course instructors and therefore only with a small sample size. Finally Case Study III and IV applied ProCeeD in rather small projects. As we observed only a small amount of participants the results should be considered anecdotal evidence.

Table 5.4: Evaluation: Summary

No.	Hypothesis	Case Study	Result
1	With ProCeeD developers adopt a common delivery process for revolutionary as well as evolutionary prototypes.	I,III,IV	Supported
1a	Using ProCeeD developers deliver revolutionary prototypes using an automated process.	I,III,IV	Supported
1b	Using ProCeeD developers deliver revolutionary prototypes using same tools and delivery process as they use for evolutionary prototypes.	I,III,IV	Supported
2	ProCeeD reduces the length of a prototyping iteration.	I	Supported
2a	Using ProCeeD developers need less time to deliver prototypes compared to developers not using ProCeeD.	I	Supported
2b	Teams who adopt ProCeeD see a shorter overall iterations length compared to Non-ProCeeD teams.	I	Supported
3	ProCeeD allows developers to collect more and better feedback.	I,III	Partially Supported
3a	Teams who use ProCeeD see more feedback than Non-ProCeeD teams.	I,III	Supported
3b	Teams who adopt ProCeeD receive better feedback than Non-ProCeeD teams.	I,III	Rejected
4	Developers see a benefit of applying ProCeeD in their workflow.	I	Supported
4a	Developers used ProCeeD to download a release.	I	Supported
4b	Developers used ProCeeD to lookup feedback.	I	Supported
4c	Developers used ProCeeD's integration with issue tracking tool and communication tools.	I	Supported
5	ProCeeD improves the traceability of software delivery processes in multi-project organizations.	II	Supported
5a	ProCeeD reduces the time developers need to monitor the delivery process.	II	Supported
5b	ProCeeD allows developers to optimize the frequency of their deliveries.	II	Supported

Prototypes developed to explore and elicit requirements are currently facing two problems. First, prototypes are created with extensive and mature tool support but are seldom delivered as executable releases to the target environment. Second, the delivery process applied for revolutionary prototypes diverges from the delivery process used for evolutionary prototypes. We found that no available prototyping approaches or tools address the delivery of revolutionary as well as evolutionary prototypes to the target environment using a common process.

6.1 Contributions

This dissertation addresses these problems with three contributions which are summarized in 6.1. The *ProCeeD Framework* allows developers to create and deliver executable prototypes to the target environment using an automated and repeatable process and provides a common feedback mechanism for users. By delivering and discussing prototypes, developers can define and refine requirements in collaboration with their stakeholders. Using ProCeeD, developers can test hypotheses on ambiguous or unclear requirements by using executable prototypes. Stakeholders can evaluate these prototypes and provide feedback to evaluate existing requirements or to make new requirements emerge.

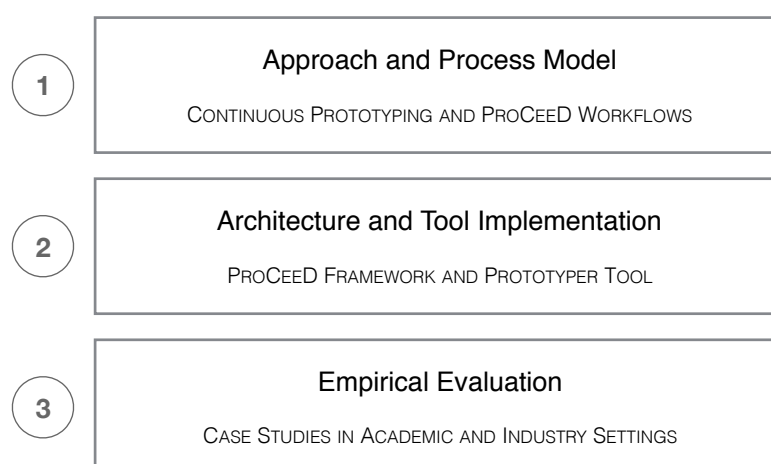


Figure 6.1: Contributions.

In addition, ProCeeD allows the delivery of several prototypes to compare alternatives in the target environment. ProCeeD is not limited to the delivery of prototypes during requirements elicitation; it establishes a common software delivery solution from requirements elicitation to software evolution. Developers can apply the same delivery and feedback process throughout the whole lifespan of a development project.

The *Prototyper* tool is based on ProCeeD and implements a common process for the delivery of executable prototypes in the field of mobile application engineering. Prototyper allows the transformation of prototypes created with prototyping tools into mobile apps which are executable in the target environment. Prototyper supports the delivery of multiple releases to a group of users. In addition, prototypes can be created with different prototyping techniques, e.g. paper prototyping and digital prototyping. Using Prototyper's feedback component, developers can collect in-situ feedback on these prototypes.

ProCeeD and Prototyper were empirically evaluated in *four case studies*. Case Study I evaluated ProCeeD's delivery workflows in two instances of a multi-project capstone course, each with more than 100 developers working in 11 industry projects. We showed that teams who adopt ProCeeD reduce the length of a prototyping iteration when compared with projects not adopting the framework. ProCeeD teams showed a two-fold increase of revolutionary prototype iterations delivered to the target environment compared to teams not applying the framework. Case Study II evaluated ProCeeD's process metrics workflow by validating the adoption of ProCeeD in project-based organizations in a capstone course across two semesters. We showed how ProCeeD helps to improve the manageability of the delivery process. Case Study III applied ProCeeD in a commercial project with a partner from the pharmaceutical industry. It was shown that ProCeeD establishes a common software delivery and feedback process for revolutionary as well as evolutionary prototypes. In addition we presented how ProCeeD and Prototyper allow the delivery of early prototypes to the end users of a system. Case Study IV introduced ProCeeD in a project using a storyboard-based requirements elicitation approach. We showed how ProCeeD can be applied to deliver storyboard-based prototypes.

6.2 Future Work

In this section we provide an outlook on future research directions and improvements regarding the ProCeeD Framework and the Prototyper tool.

Further Evaluation

We have evaluated ProCeeD and Prototyper in multiple case studies in both academic and commercial contexts. The projects were carried out by 6-8 developers each, lasted for three months and involved small groups of users. Future evaluations could assess the following aspects of ProCeeD: it could be evaluated how ProCeeD can be adopted in projects with a large amount of users. This could raise additional challenges in the areas of, e.g. expectation and feedback management. Furthermore, it should be analyzed how ProCeeD performs when multiple target environments need to be supported at the same time. This can induce overhead due to additional complexity in the software development and delivery process. Finally, the impact of ProCeeD on the communication within software teams could be explored. While Case Study III showed that ProCeeD's feedback component was frequently used, the impact of an automated delivery process for prototypes on the communication between users, interaction designers, developers and clients needs to be further evaluated.

Prototype Exchange Format

In Chapter 2 we showed that various commercial and non-commercial tools for creating revolutionary prototypes are available. Each of the tools builds on its own proprietary data exchange format to describe and store prototypes. In ProCeeD we demonstrated how to transform and package such prototypes into mobile apps which can be executed in different target environments. We currently need to implement the necessary transformation steps for each prototyping tool separately. A common intermediary format to store and interchange prototypes would solve this problem. Defining and publishing such a format could facilitate adoption among the prototyping tool vendors and simplify the application of the ProCeeD workflows in software projects.

Extension of ProCeeD and Prototyper

The integration of Prototyper into the development tool chain could be further improved. As chat-based control ("ChatOps") of software delivery processes gains importance in industry, Prototyper could be extended by a chat-based user interface. The interface could not only allow developers to deliver and promote releases but also to manage and even react to user feedback instantly and without leaving the communication tool of their choice.

To simplify the creation of hybrid prototypes we could extend Prototyper's integration into IDEs. Developers currently need to go through the manual step of downloading and integrating a revolutionary prototype after it was altered in the respective prototyping tool, which could be managed from within the IDE.

While Prototyper is capable of delivering multiple, alternative releases to a group of users at the same time, its feedback management capabilities do not yet allow developers to look up if a user had access to one or multiple releases while he provided a certain feedback item. This is not a problem in projects with a small amount of users, as developers can structure the feedback manually, but it is important to improve on this to allow the usage of Prototyper in projects with a large amount of users and in order to conduct A/B tests with early prototypes.

Finally, ProCeeD's user feedback model is another opportunity for future work. ProCeeD's current feedback model is release-centric, each feedback item is collected and stored for a specific release and a specific part of the user interface. Especially during the early prototyping of a feature, users may come up with visionary ideas which will question e.g. more than just a part of the user interaction model. For this we could extend ProCeeD's feedback model to accommodate more general feedback and provide developers with the necessary workflows to process it [DKAB16].

With the Prototyper tool we developed a software delivery solution for mobile applications which allows to continuously deliver revolutionary and evolutionary prototypes using a common workflow. In the future we plan to publish Prototyper both as open source software and using a Software-as-a-Service approach. We believe that applying Continuous Prototyping using ProCeeD and Prototyper will have a lasting benefit on the collaboration between interaction designers, developers and users and we are looking forward to where the journey takes us from here!

- [ADB16] L. Alperowitz, D. Dzvonyar, and B. Bruegge. "Metrics in Agile project courses." In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 323–326.
- [Alp+17] L. Alperowitz, Weintraud, A. Marie, K. S. Christoph, and B. Bruegge. "Continuous Prototyping." In: *3rd International Workshop on Rapid Continuous Software Engineering (RCoSE'17)*. ACM. Buenos Aires - Argentina, 2017.
- [And89] S. J. Andriole. *Storyboard prototyping: a new approach to user requirements analysis*. QED Information Sciences, Inc., 1989.
- [ASF17] L. Alperowitz, C. Scheuermann, and N. Frankenberg. "From Storyboards to Code: Visual Product Backlogs in Agile Project Courses." In: *15. Workshop Software Engineering im Unterricht der Hochschulen (SEUH)*. 2017.
- [BB01] B. Boehm and V. R. Basili. "Top 10 list [software development]." In: *Computer* 34.1 (2001), pp. 135–137.
- [BBLZ96] D. Bäumer, W. R. Bischofberger, H. Lichter, and H. Züllighoven. "User interface prototyping—concepts, tools, and experience." In: *Proceedings of the 18th international conference on Software engineering*. IEEE, 1996, pp. 532–541.
- [BD09] B. Bruegge and A. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns and Java*. Prentice Hall, 2009.
- [Bec+01] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. "Manifesto for agile software development." In: (2001).
- [Bec00] K. Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.
- [BKA15] B. Bruegge, S. Krusche, and L. Alperowitz. "Software engineering project courses with industrial clients." In: *ACM Transactions on Computing Education (TOCE)* 15.4 (2015), p. 17.
- [BKKZ92] R. Budde, K. Kautz, K. Kuhlenkamp, and H. Züllighoven. *Prototyping*. Springer, 1992.

- [Boe88] B. W. Boehm. "A spiral model of software development and enhancement." In: *Computer* 21.5 (1988), pp. 61–72.
- [Boo91] G. Booch. *Object Oriented Design with Applications*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1991.
- [BPKR09] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer. *Software & Systems Requirements Engineering: In Practice*. McGraw-Hill Education, 2009.
- [Car00] J. M. Carroll. *Making use: scenario-based design of human-computer interactions*. MIT press, 2000.
- [CLL03] C. K. Chua, K. F. Leong, and C. S. Lim. *Rapid Prototyping: Principles and Applications, 2nd Edition*. World Scientific Publishing Co Inc, 2003.
- [COB06] O. Creighton, M. Ott, and B. Bruegge. "Software cinema-video-based requirements engineering." In: *Requirements Engineering, 14th International Conference*. IEEE. 2006, pp. 109–118.
- [Con93] S. A. Conger. *The new software engineering*. Course Technology Press, 1993.
- [Coo99] A. Cooper. "The Inmates are Running the Asylum—Why High-Tech Products Drive Us Crazy and How 2 to Restore the Sanity." In: *SAMS* (1999).
- [Dav92] A. M. Davis. "Operational prototyping: A new development approach." In: *IEEE software* 9.5 (1992), pp. 70–78.
- [DeM79] T. DeMarco. "Structure analysis and system specification." In: *Pioneers and Their Contributions to Software Engineering*. Springer, 1979, pp. 255–288.
- [Dij79] E. W. Dijkstra. *Panel remarks*. 4th International Conference on Software Engineering. 1979.
- [DKA14] D. Dzvonyar, S. Krusche, and L. Alperowitz. "Real projects with informal models." In: *MODELS'14 - Proceedings of the 10th Educators' Symposium*. 2014.
- [DKAB16] D. Dzvonyar, S. Krusche, R. Alkadhi, and B. Bruegge. "Context-aware user feedback in continuous software evolution." In: *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. ACM. 2016, pp. 12–18.
- [DMG07] P. M. Duvall, S Matyas, and A Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.
- [ESSD08] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. "Selecting Empirical Methods for Software Engineering Research." In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. London: Springer, 2008, pp. 285–311.

-
- [FF06] M. Fowler and M. Foemmel. "Continuous integration." In: *Thought-Works* (<https://martinfowler.com/articles/continuousIntegration.html>) (2006), p. 122.
- [Flo84] C. Floyd. "A systematic look at prototyping." In: *Approaches to prototyping*. Springer, 1984, pp. 1–18.
- [FN00] N. E. Fenton and M. Neil. "Software metrics: roadmap." In: *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 357–370.
- [Fre10] R. E. Freeman. *Strategic management: A stakeholder approach*. Cambridge University Press, 2010.
- [GGSF14] R. Ganhör, F. Güldenpfennig, O. Subasi, and G. Fitzpatrick. "Towards Fast and Interactive Prototypes of Mobile Apps." In: *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design*. OzCHI '14. USA: ACM, 2014, pp. 328–331.
- [GL85] J. D. Gould and C. Lewis. "Designing for usability: key principles and what designers think." In: *Communications of the ACM* 28.3 (1985), pp. 300–311.
- [GLZ99] G. Guida, G. Lamperti, and M. Zanella. "Software Prototyping in Data and Knowledge Engineering." In: Springer, 1999. Chap. The Prototyping Approach to Software Development, pp. 1–32.
- [Gra94a] R. B. Grady. "Successfully applying software metrics." In: *Computer* 27.9 (1994), pp. 18–25.
- [Gra94b] I. Graham. *Object Oriented Methods, 2nd Edition*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.
- [GW07] M. Glinz and R. J. Wieringa. "Stakeholders in requirements engineering." In: *IEEE software* 24.2 (2007), pp. 18–20.
- [Hau98] J. Hauschildt. *Promotoren: Champions der Innovation*. Gabler Verlag, 1998.
- [HB95] K. Holtzblatt and H. R. Beyer. "HumanFACTOR." In: *Communications of the ACM* 38.5 (1995), p. 31.
- [HDK93] P. Hsia, A. M. Davis, and D. C. Kung. "Status report: requirements engineering." In: *IEEE* 10.6 (1993), pp. 75–79.
- [HF10] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.
- [HL01] H. F. Hofmann and F. Lehner. "Requirements engineering as a success factor in software projects." In: *IEEE software* 18.4 (2001), p. 58.

- [HL13] V. Hoffmann and H. Lichter. *Rapid prototyping in der use-case-zentrierten Anforderungsanalyse*. Tech. rep. Fachgruppe Informatik, 2013.
- [IBM81] IBM. *IBM Archives: Announcement Press Release IBM PC*. 1981. URL: https://www-03.ibm.com/ibm/history/exhibits/pc25/pc25_press.html.
- [Jac+99] I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, and G. Booch. *The unified software development process*. Vol. 1. Addison-wesley Reading, 1999.
- [Jac93] I. Jacobson. *Object-oriented software engineering: a use case driven approach*. Pearson Education, 1993.
- [JCK10] A. P. Jørgensen, M. Collard, and C. Koch. "Prototyping iPhone Apps: Realistic Experiences on the Device." In: *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*. NordiCHI '10. New York, NY, USA: ACM, 2010, pp. 687–690.
- [KA14] S. Krusche and L. Alperowitz. "Introduction of continuous delivery in multi-customer project courses." In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 335–343.
- [KABW14] S. Krusche, L. Alperowitz, B. Bruegge, and M. O. Wagner. "Rugby: an agile process model based on continuous delivery." In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. ACM. 2014, pp. 42–50.
- [Kan+04] C. Kaner et al. "Software engineering metrics: What do they measure and how do we know?" In: *Proceedings of METRICS'04*. 2004.
- [Kel84] J. F. Kelley. "An iterative design methodology for user-friendly natural language office information applications." In: *ACM Transactions on Information Systems (TOIS)* 2.1 (1984), pp. 26–41.
- [KHSI12] B. Köhler, J. Haladjian, B. Simeonova, and D. Ismailović. "Feedback in low vs. high fidelity visuals for game prototypes." In: *Proceedings of the Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques*. IEEE, 2012, pp. 42–47.
- [KKLK05] S. Kujala, M. Kauppinen, L. Lehtola, and T. Kojo. "The role of user involvement in requirements quality and project success." In: *13th IEEE International Conference on Requirements Engineering (RE'05)*. 2005, pp. 75–84.
- [Kle+15] S. Klepper, S. Krusche, S. Peters, B. Bruegge, and L. Alperowitz. "Introducing continuous delivery of mobile apps in a corporate environment: a case study." In: *2nd International Workshop on Rapid Continuous Software Engineering (RCoSE)*. IEEE/ACM. 2015, pp. 5–11.

-
- [Kru04] P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.
- [Kru16] S. Krusche. "Rugby - A Process Model for Continuous Software Engineering." PhD thesis. Technical University Munich, Germany, 2016.
- [KS98] G. Kotonya and I. Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.
- [Kuj03] S. Kujala. "User involvement: a review of the benefits and challenges." In: *Behaviour & information technology* 22.1 (2003), pp. 1–16.
- [Lan96] J. A. Landay. "SILK: sketching interfaces like crazy." In: *Conference companion on Human factors in computing systems*. ACM. 1996, pp. 398–399.
- [Las+15] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, and M. S. Bernstein. "Apparition: Crowdsourced User Interfaces That Come to Life As You Sketch Them." In: *Proceedings of the 33rd Annual Conference on Human Factors in Computing Systems*. CHI '15. USA: ACM, 2015, pp. 1925–1934.
- [LDL98] A. van Lamsweerde, R. Darimont, and E. Letier. "Managing conflicts in goal-driven requirements engineering." In: *IEEE Transactions on Software Engineering* 24.11 (1998), pp. 908–926.
- [Mac93] L. Macaulay. "Requirements capture as a cooperative activity." In: *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. IEEE. 1993, pp. 174–181.
- [Mai98] N. A. M. Maiden. "CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements." In: *Domain Modelling for Interactive Systems Design*. Ed. by A. Sutcliffe and D. Benyon. USA: Springer, 1998, pp. 39–66.
- [MP84] S. M. McMenamin and J. F. Palmer. *Essential systems analysis*. Yourdon Press, 1984.
- [Nar15] S. Narayan. *Agile IT Organization Design: For Digital Transformation and Continuous Delivery*. Pearson Education, 2015.
- [ND86] D. A. Norman and S. W. Draper. *User centered system design*. L. Erlbaum Associates Inc., 1986.
- [NE00] B. Nuseibeh and S. Easterbrook. "Requirements Engineering: A Roadmap." In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46.
- [Nie94] J. Nielsen. *Usability Engineering*. Elsevier, 1994.
- [Nor12] D. Norman. *The design of everyday things*. Basic Books, 2012.
-

- [OAB12] H. H. Olsson, H. Alahyari, and J. Bosch. "Climbing the "Stairway to Heaven"-A Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software." In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2012, pp. 392–399.
- [Pag13] D. Pagano. *PORTNEUF - A Framework for Continuous User Involvement*. Verlag Dr. Hut, 2013.
- [PEM03] F. Paetsch, A. Eberlein, and F. Maurer. "Requirements Engineering and Agile Software Development." In: *Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. WETICE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 308–.
- [Pet16] S. M. Peters. "MIBO – A Framework for the Integration of Multimodal Intuitive Controls in Smart Buildings." Dissertation. München: Technische Universität München, 2016.
- [Poh10] K. Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. 1st. Springer, 2010.
- [RC09] M. B. Rosson and J. M. Carroll. "Scenario based design." In: *Human-computer interaction*. Boca Raton, FL (2009), pp. 145–162.
- [Roy+70] W. W. Royce et al. "Managing the development of large software systems." In: *proceedings of IEEE WESCON*. Vol. 26. 8. Los Angeles. 1970, pp. 1–9.
- [RSI96] J. Rudd, K. Stern, and S. Isensee. "Low vs. high-fidelity prototyping debate." In: *Interactions* 3.1 (1996), pp. 76–85.
- [Rup+07] C. Rupp et al. "Requirements-Engineering und-Management: Professionelle, iterative Anforderungsanalyse für die Praxis. 4." In: *Aufl. Hanser, München* (2007).
- [SB02] K. Schwaber and M. Beedle. "Agile Software Development with Scrum." In: (2002).
- [Sny03] C. Snyder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, 2003.
- [SSBV16] C. Scheuermann, M. Strobel, B. Bruegge, and S. Verclas. "Increasing the Support to Humans in Factory Environments using a Smart Glove: An Evaluation." In: *2016 International Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*. Toulouse, France: IEEE, 2016, pp. 847–854.
- [Sta+12] H. Stangl et al. "SCRIPT: A Framework for Scenario-Driven Prototyping." PhD thesis. Technische Universität München, 2012.

-
- [STG03] R. Sefelin, M. Tscheligi, and V. Giller. "Paper prototyping-what is it good for?: a comparison of paper-and computer-based low-fidelity prototyping." In: *CHI'03 extended abstracts on Human factors in computing systems*. ACM. 2003, pp. 778–779.
- [TB90] S. D. Tripp and B. Bichelmeyer. "Rapid prototyping: An alternative instructional design strategy." In: *Educational Technology Research and Development* 38.1 (1990), pp. 31–44.
- [Urb92] J. E. Urban. *Software Prototyping and Requirements Engineering*. Tech. rep. Arizona State University, 1992.
- [Ver89] L. Vertelney. "Using video to prototype user interfaces." In: *ACM SIGCHI Bulletin* 21.2 (1989), pp. 57–61.
- [Vir89] R. A. Virzi. "What can you learn from a low-fidelity prototype?" In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 33. 4. SAGE Publications. 1989, pp. 224–228.
- [VT91] J. M. Vlissides and S. Tang. "A unidraw-based user interface builder." In: *Proceedings of the 4th annual ACM symposium on User interface software and technology*. ACM. 1991, pp. 201–210.
- [Wit73] E. Witte. *Organisation für Investitionsentscheidungen - Das Promotorenmodell*. 1973.
- [WTL02] M. Walker, L. Takayama, and J. A. Landay. "High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 46. 5. SAGE Publications. 2002, pp. 661–665.