Real Projects with Informal Models

Dora Dzvonyar, Stephan Krusche, and Lukas Alperowitz

Technische Universität München, Munich Germany
{dzvonyar,krusche,alperowi}@in.tum.de

Abstract. Informal models help to generate a shared understanding about the software to be developed as early as possible. In this paper we describe how we use informal models to react to different initial situations for project teams in our capstone course. We explain how we handle three cases ranging from projects without requirements to projects with detailed specifications. Project participants are convinced of the advantages of informal models and agree that they improve the outcome of their projects.

Keywords: Teaching, Real Projects, Informal Models, Communication Models, Prototyping, Release Management, Software Theater, Trailer

1 Introduction

In software engineering practical courses, we encourage the use of informal models such as trailers, mock-ups, scenarios and early prototypes. As opposed to formal models, informal models can be incomplete, ambiguous, or even unrealizable. We also call them communication models as they facilitate the exchange of ideas between customers and developers [1].

In this paper we describe how we embed informal models into the lifecycle of our project course and explain how we teach their usage to students. In particular, we show three examples in which informal models help to deal with different situations in the beginning of a project.

2 Course Description

In summer 2014, we conducted a multi-customer capstone course called iOS Praktikum in which students developed mobile applications with industry partners and real deadlines. A project-based organization allows us to run multiple software engineering projects at the same time [1]. 100 participants developed 16 applications for 11 industry partners during the 2014 summer term. The results of the course, including the video recordings of the final presentations, are available on the course website.[1] In this course we took the roles of a Team Coach (Dora Dzvonyar) and Instructors (Stephan Krusche and Lukas Alperowitz). For a detailed description of those roles see [1].

[1] http://www1.in.tum.de/ios14

We organize the projects in an agile way using Scrum [2] and elements of the Unified Process [3]. The overall timeline is shown in fig. 1. Our course runs over a period of three months and starts with a kickoff event in which each customer presents their problem. The students then solve these problems in teams and showcase their progress in two course-wide presentations, the Design Review and the Customer Acceptance Test.

In the Design Review, which takes place after two thirds of the course duration, they focus on presenting the overall architecture together with the rationale behind their design decisions. They also show a trailer which they typically create in the beginning of the project. The technique of using video for requirements elicitation, called software cinema [4], has shown to help the participants to communicate the main idea of their project. At the end of the course, we organize a Customer Acceptance Test (CAT) in which each team presents their final application. In both presentations, especially in the CAT, we encourage the participants to demonstrate their product by acting out a scenario while showcasing their application. We call this technique software theater and believe that it enriches the presentations by giving a more comprehensible impression of the project outcome [5].

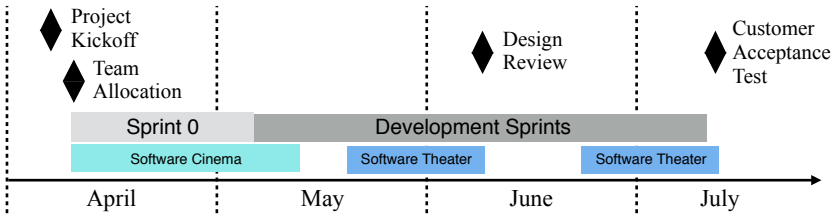

Fig. 1. Capstone course timeline

The prior knowledge of participants varies from undergraduates in their first year up to graduates with profound experience in software engineering. During the first week of the course, we assign the students to project teams based on their personal preferences as well as their prior experience with the goal of creating balanced teams. This approach permits less experienced students to learn from experienced ones and facilitates knowledge exchange within the project team [6]. Additionally we hold interactive tutorials in which we familiarize the students with the tools for creating informal models. Different velocities in the teams lead to the situation, that some participants gain first experiences with informal models even before we cover the topic in the tutorial. Therefore we use concepts from experiential learning [7], encouraging them to build best practices and learn from their experiences [8].

Most of the teams receive a problem statement from their customer right at the beginning of the course. It includes at least one visionary scenario describing the purpose of the system [9]. To build team spirit and to synchronize all students

regarding technical expertise as well as understanding of the requirements, each team conducts a non-development sprint, which we call *Sprint 0* [1]. In this sprint we encourage the use of informal models to create a common understanding of the problem between team and customer.

After Sprint 0, the students work with these informal models, transforming them into executable prototypes in regular development sprints which usually last one to two weeks. In the later stages of the course, they refine and formalize their early models, but also create new informal models, e.g. to quickly propose a solution to a change request made by the customer. When the team wants to obtain feedback, the students deliver an executable prototype to their customer using the continuous delivery workflow described in [10].

In recent years, we conducted evaluations using questionnaires about how informal models, in particular executable prototypes, are used by the project teams and in management meetings throughout the whole course lifecycle. Results of our evaluations can be found in [8] and [10]. In the following section, we concentrate on the use of various informal models during the initial phase of the projects.

3 Case Study

The heterogeneity of projects and customers in our course leads to the situation, that some teams start with a detailed problem statement with concrete requirements, while others begin their project only with a vague idea. This results in varying starting positions and differences in team velocities which raise challenges at the beginning of the course.

Depending on the level of detail of the problem statement, the teams lay a different focus on the three activities requirements elicitation, analysis and design in Sprint 0. This section describes three typical cases of initial positions and outlines how the teams use informal models to master their individual situation. These situations are shown in fig. 2 together with the lifecycle activities the teams need to focus on and the required artifacts as black diamonds, that also represent important milestones in the lifecycle model. In each case we report about the opinions of one project participant regarding the use of informal models.

Situation 1: There is no problem statement and no top-level design

If the team does not receive a problem statement, the developers focus on requirements elicitation activities during the first weeks of the project. Projects without problem statements usually involve a high degree of innovation, and sometimes the customer does not have a clear idea of the desired application in the beginning. During requirements elicitation, developers and customer elaborate and agree on at least one visionary scenario describing the main purpose of the application. This scenario then serves as basis for analysis and design.

In the summer 2014 course, one of the customers without a problem statement was Siemens IT. The team was presented with a vague idea of an application motivating employees to get to know a company-internal reporting solution.

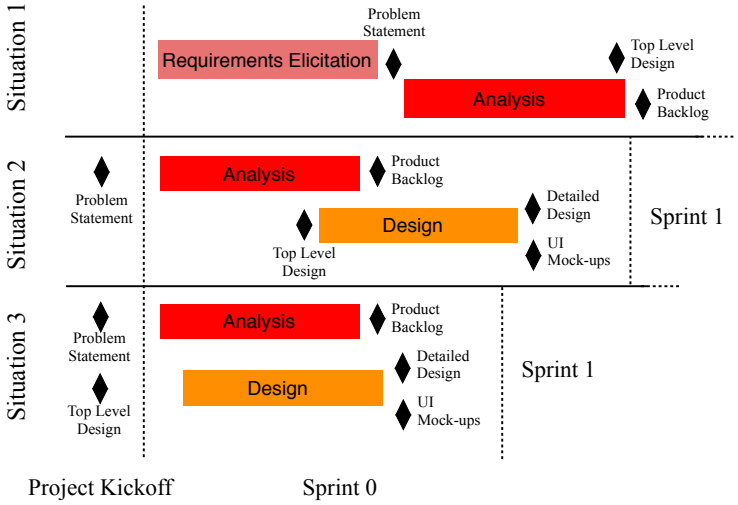

Fig. 2. Varying initial situations, based on the lifecycle model described in [10]

Thiemo Taube, the team coach, guided the developers through the requirements elicitation phase. According to him, creating a short trailer for the application helped the developers to generate a common understanding of the purpose of their product. The team members also wrote visionary scenarios describing the way users interact with the system. These informal modeling activities motivated the developers to analyze all aspects of the problem and to discuss and verify their assumptions with the customer.

Situation 2: There is a problem statement, but no top-level design

If the team receives a problem statement including a detailed visionary scenario, it can immediately start to divide the requirements into backlog items, e.g. user stories. The developers can also define the top-level design, describing the overall architecture and potential technologies used for the communication between subsystems. Both artifacts provide the basis for the following design activities, in which the team members refine the top-level design. The result of these activities is a detailed subsystem decomposition which they visualize in an UML Deployment Diagram [9].

Furthermore, they start designing the user interface of the application. We encourage the developers to use low-fidelity mock-ups, e.g. drawn on paper or created using a tool such as Balsamiq [11]. One advantage of unpolished interface designs is that they are fast to produce and easy to adapt [12]. Thus, they are well-suited for facilitating discussions between stakeholders early in the project. Moreover, showing low-fidelity mock-ups to the customer helps to avoid unrealistic expectations regarding the status of the implementation. If the team produces perfectly refined screen designs from the beginning, they risk

the customer thinking that the application is almost finished even though the actual functionality is not yet implemented [13]. Research has also shown that unpolished designs generate more feedback than high-fidelity ones because the customer is not afraid that his comments lead to high modification effort [14].

In the BMW project, the team obtained a detailed problem statement. The goal was to design an application for conveniently arranging appointments between drivers and car workshops. Vitus Holzner, customer of the project, reports that receiving early prototypes was very useful both during Sprint 0 and the regular development sprints. According to him, they helped to detect misunderstandings and ensured that developers understood the complex details of the problem statement. He was able to get an impression of how it feels to use the application instead of having to imagine it based on a formal specification.

Situation 3: There is a problem statement and a top-level design

If the team receives a detailed problem statement and a top-level design from the customer, both analysis and design activities can start immediately. This is usually the case when developers are refining an existing application or the customer has profound background knowledge in software engineering.

An example in this category was the NTT DATA project. The team improved the user interface of an already existing iPhone app for finding and reserving electric bikes. It also extended the application with the capability of reporting damages on a defective bike. The customer of the project, Frank von Eitzen, recommends that project participants communicate their ideas of a system through informal models whenever possible, not only during requirements elicitation. Fig. 3 shows a top-level design which he used in the kickoff presentation. While it does not follow any formal rules, it helps to convey the overall system architecture. It is also easier to understand than e.g. an UML Deployment Diagram.

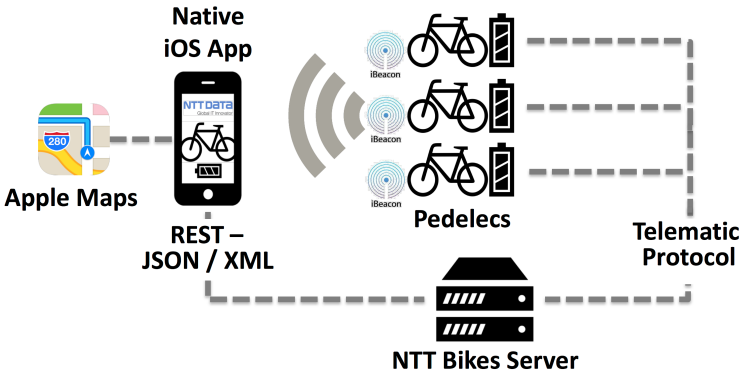

Fig. 3. Example of an informal top-level design

According to Frank von Eitzen, low-fidelity interface designs such as pen and paper sketches are very useful informal models. They facilitate discussions while keeping both customers and developers focused on the essential parts of the application. They were used in the project to choose between different design alternatives for a particular functionality. Fig. 4 shows from left to right the evolution of a first whiteboard sketch of a new screen through a mock-up made with Balsamiq into a finished interface. He also reports that executable prototypes delivered by the development team gave him a good insight into the status of the implementation. They allowed him to verify that the agreed functionality had been realized as desired and to plan the subsequent iteration.

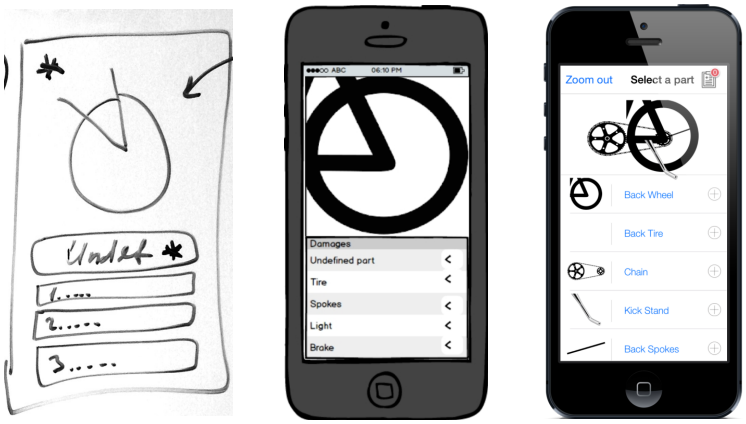

Fig. 4. Evolution of the user interface, from rough sketch (left) to the delivered application (right)

4 Conclusion

In this paper we described how we use informal models in our capstone course. In three exemplary projects, we showed how informal models help students to deal with different starting situations. While factors such as the degree of innovation or the technical requirements of a project influence team velocity, the initial situation of a team is determined largely by the level of specification provided by the customer. For each case, we outlined how informal models helped them to master these individual challenges.

We also described how informal models can help to generate a shared understanding in the early stages of software development. They facilitate communication and discussion, and encourage customers to give feedback. Our project participants are convinced of the advantages of informal modeling and agree that it has improved the outcome of their projects. While we encourage our

students to use informal models we also teach them how to transition to formal specification at later stages in the course.

References

1. Bruegge, B., Krusche, S., Wagner, M.: Teaching Tornado. In: Proceedings of EduSymp '12, ACM (2012) 5–12
2. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall PTR (2002)
3. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)
4. Creighton, O., Ott, M., Bruegge, B.: Software cinema - video-based requirements engineering. In: Requirements Engineering, 14th IEEE International Conference, IEEE (2006) 109–118
5. Bruegge, B., Krusche, S., Alperowitz, L.: Software Engineering Project Courses with Industrial Clients. ACM Transactions on Computing Education (TOCE) (2015)
6. Braun, A., Dutoit, A. Bruegge, B.: A software architecture for knowledge acquisition and retrieval for global distributed teams. In: GSD'03 The International Workshop on Global Software Development. (2003) 24
7. Kolb, D.: Experiential learning: Experience as the source of learning and development. Prentice Hall. (1984)
8. Krusche, S., Alperowitz, L.: Introduction of Continuous Delivery in Multi-customer Project Courses. In: Proceedings of ICSE'14, ACM (2014) 335–343
9. Bruegge, B., Dutoit, A.H.: Object Oriented Software Engineering Using UML, Patterns, and Java. Prentice Hall International (2009)
10. Krusche, S., Alperowitz, L., Bruegge, B., Wagner, M.: Rugby: An Agile Process Model Based on Continuous Delivery. In: Proceedings of RCoSE'14, ACM (2014) 42–50
11. Balsamiq Studios: Balsamiq Mockups (2014) `http://balsamiq.com/products/mockups`, accessed 07/18/2014.
12. Mayhew, D.: The Usability Engineering Lifecycle: A Practitioner's Guide to User Interface Design. Morgan Kaufmann Publishers (1999)
13. Spolsky, J.: The Iceberg Secret, Revealed. In: Joel on Software. Springer (2004) 189–195
14. Rudd, J., Stern, K., Isensee, S.: Low vs. high-fidelity prototyping debate. Interactions (1996) 76–85