Modeling in Agile Project Courses

Lukas Alperowitz*, Jan Ole Johanssen†, Dora Dzvonyar‡, and Bernd Bruegge§
Department of Informatics, Technical University of Munich, Germany
*lukas.alperowitz@tum.de, †jan.johanssen@tum.de, ‡dora.dzvoynar@tum.de, §bruegge@in.tum.de

***Abstract*—Teaching software engineering in an applied setting with projects provided by clients from industry creates a real-world learning experience for students. While clients are domain experts well-aware of the system's requirements, they often lack technical knowledge required to make decisions regarding the system architecture or the technologies involved in the project. Therefore, it is challenging for students to maintain a common language of understanding.**

To overcome this obstacle, we suggest a four-perspective modeling approach that allows students to create models as a basis for communicating with clients that don't have a technical background. The student teams apply these models to clarify requirements as well as to discuss architectural considerations and technological decisions. Each modeling perspective focuses on different aspects of the system to be developed. We teach this approach using a methodology that integrates modeling into agile software engineering project courses.

I. Introduction

Applied software engineering project courses should involve industry partners and their real-world problems to provide more relevance and prepare students for their later careers [2], [6], [9]. We regularly conduct such courses with 100 students developing applications in 10-12 projects in which we refer to the industry partners as *clients* [5]. While universities aim to teach their students in a real-world setting, industry partners want to benefit in terms of creative uses of technology and innovative product ideas. Therefore, the role of the client in such a course goes beyond stating requirements upfront at the beginning of a project; they need to understand and take decisions regarding technical details of the future system throughout the whole lifespan of the project.

In our course, the clients are often domain experts who are highly familiar with the problem at hand, but they rarely possess knowledge that is necessary to understand different implementation-specific alternatives, nor to express and communicate their needs concerning technological aspects of the project. The lack of a common basis of communication poses a challenge for students to discuss architectural considerations and technical details with their clients.

In this paper, we describe a four-perspective modeling approach which we use to overcome this obstacle. Aside from being easily learnable for students, its main goal is to establish a common terminology which is utilized for discussions with clients who are not familiar with software modeling languages such as UML. Each perspective of the modeling approach puts a different aspect of the system into focus and presents it using an appropriate level of abstraction.

The remainder of this paper is structured as follows. In Section II, we outline related work. Section III introduces the four-perspective modeling approach by describing the models, their perspective on the system at hand, and the transition between each of these models. We provide an approach for teaching the models in Section IV. Advantages and disadvantages are discussed in Section V. Section VI concludes the paper and describes starting points for future work.

II. Related Work

Lemma et al. present a tablet-based approach utilizing touch gestures for creating and managing object-oriented software models [8]. Wüest et al. propose a similar tablet-based approach, allowing multiple developers to collaborate on thoughts and ideas while deviating from standards such as UML [11]. Both approaches demonstrate the benefits of informal modeling, but they focus on software engineers as the target user group. To the best of our knowledge, an educational method for informal models of software systems that allows students in agile projects to communicate their ideas and thoughts to a non-technical audience has not yet been addressed in previous research.

URML, a language to model requirements knowledge relies on visual notation to make models understandable by various stakeholders and extends UML in its capability to describe the complex requirements of a system [10]. URML replaces some UML class boxes with icons to "enhance the vividness of the diagrams" [3]. With our four-perspective approach, we adopt the idea of replacing elements of formal models with visual representations and extend it to more diagram types.

In our previous work, we presented how we use informal models in our capstone course [7]. The four-perspective modeling approach builds upon this work, further structures our teaching method, and describes our latest experiences.

III. Four-Perspective Modeling

In this section, we describe our approach in detail by first explaining the rationale behind each model and specifying its target audience. Moreover, we give a brief overview of the benefits and weaknesses of each model and discuss when it is best applied to communicate a particular perspective of a software system. We also explain how transitions between the models support their understanding and thereby facilitate their communication. Several of the models of our four-perspective approach are described in detail in [4].

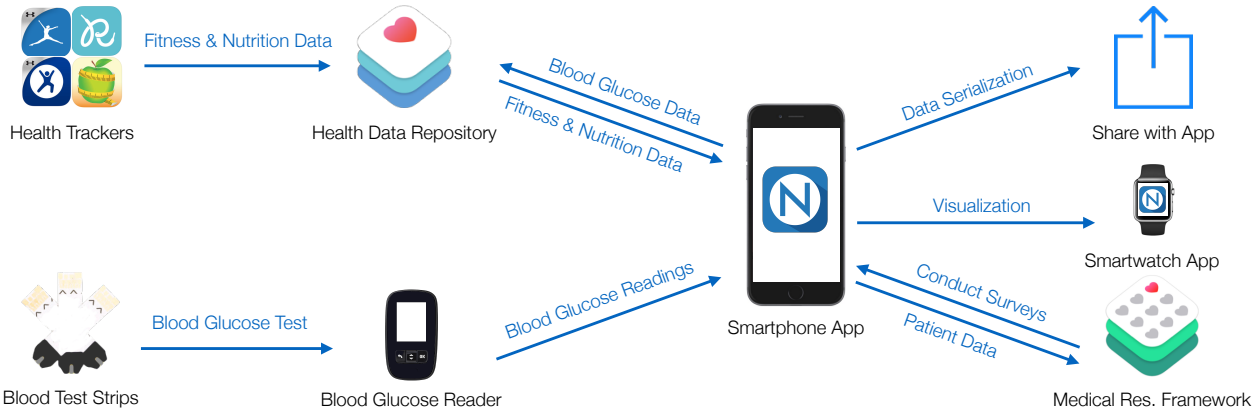

Fig. 1: Top-Level Design of a system for mobile blood glucose level tracking.

To illustrate the different perspectives, we describe a system that allows users to track their blood glucose level using their smartphone in a mobile context[1].

A. Top-Level Design

The *Top-Level Design* expresses the overall architecture of a system by highlighting its subsystems and the data exchanged between them as shown in Fig. 1. The notation does not follow strict rules, but conforms to conventions: each subsystem is depicted with an icon and labeled with a textual description. Arrows and a short textual description describe the data that is exchanged between the subsystems. Due to its limited complexity, a Top-Level Design is especially useful to explain and discuss software architectures with non-technical stakeholders of a software project.

The Top-Level Design models the problem domain for technology-independent discussions to keep the focus on the system's overall architecture [1]. Therefore, our example in Fig. 1 shows a *Smartphone App* rather than an *Apple iPhone*, *Calorie Trackers* instead of concrete application names and a *Health Data Repository* without mentioning its actual implementation. Determining the right level of abstraction is key when creating a Top-Level Design. During the creation of a Top-Level Design, students should focus on answering the following two questions:

- *What parts of the system are relevant for the client?*
- *What can we omit to avoid unnecessary complexity and distraction?*

B. Analysis Object Model

The *Analysis Object Model* describes the static structure of the system: objects with attributes and their associations. As illustrated in the example in Fig. 2, the notation of the Analysis Object Model is similar to an iconized version of a conceptual UML class diagram. An Analysis Object Model supports the establishment of a common terminology. This is relevant when discussing parts of the system within the development team or with non-technical stakeholders.

[1]The icons used for the models are retrieved from http://www.flaticon.com and are licensed under the Creative Commons BY 3.0 license.

The Analysis Object Model should not cover every entity of the system: it depicts a high-level overview of the problem domain instead of the exact objects used for implementation. Adding visionary entities, in particular entities that are never implemented but are common in the system's problem domain, can support the model's comprehension. The same holds true for the relationships between objects: they are modeled only if they provide further guidance for the audience. With an Analysis Object Model, the students should address the following questions:

- *What are the system's core entities and their relationships?*
- *How can we align them with the client's domain of knowledge?*

C. Subsystem Decomposition

For providing a more technical overview of the system that should be developed, we leverage the notation of a UML component diagram. The *Subsystem Decomposition* offers a view beyond the subsystems previously introduced in the Top-Level Design in terms of components and their interfaces. Similar to the previously described models, the key when creating or extending this diagram is to find the right level of abstraction by visualizing the key interaction points between the subsystems and their environment.

In order to make the model understandable for a non-technical audience, students start by showing an iconized version as visualized on the right side of Fig. 3. Then, the inner structure of each subsystem on a component level is revealed step by step. Finally, the entities of the Analysis Object Model can be used to explain the data flow. This becomes more expressive when the data flow is animated with colored lines, serving as an entity's *trace* through the system. The Subsystem Decomposition allows students to provide details on components and answer the following questions:

- *How are objects processed by the system's individual components?*
- *How does the data flow between the subsystems and components?*

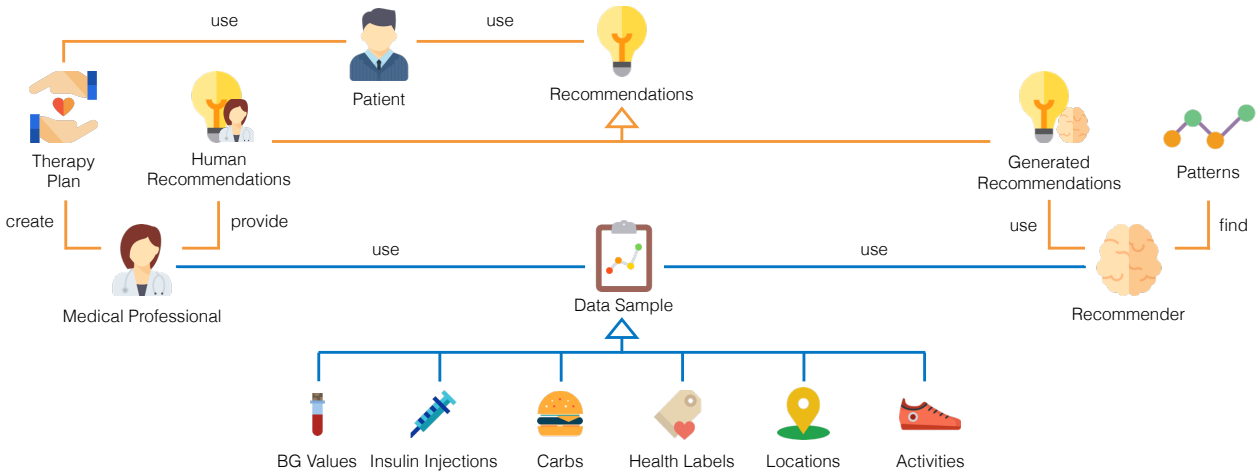

Fig. 2: Iconized Analysis Object Model visualizing important entities.

D. Hardware-Software Mapping

The *Hardware-Software Mapping* shows the distribution of a system's components on actual hardware and software nodes. As opposed to the other perspectives, the use of concrete instances and solution domain terminology is encouraged here. For instance, *iOS* is proposed as an actual mobile operating system, or *iPhone* as an implementation of a smartphone.

We recommend using a UML deployment diagram for the Hardware-Software Mapping, while still working with icons to preserve the model's understandability for a non-technical audience. The previously introduced components in the Subsystem Decomposition can be detached from the overall diagram and placed within the actual hardware components, as shown in the presentation recording of the blood glucose tracking system[2]. Using this perspective, the students can answer the following question:

- *On which hardware or software node are the components deployed on?*
- *What are the specific protocols used for communication between the nodes?*

IV. Teaching Approach

We teach our four-perspective approach with a methodology involving multiple roles and feedback rounds and concentrates not only on the *creation* of models for each perspective, but also on their effective *communication*. In this section, we explain how we teach the approach in the iPraktikum, a multi-project capstone course conducted at the Technical University of Munich. In this course, 100 students develop systems in the context of mobile applications over the course of one semester based on an agile process model called Rugby optimized for part-time developers [5].

Each project team works with a client from industry and is led by a project leader who is a doctoral candidate. Furthermore, the organizational structure includes cross-project roles which cover topics relevant for all projects, such as *modeling* or *release management*.

At the beginning of the course, each project team receives a problem statement describing the purpose and requirements of the system from their client. We precede the development sprints with a period of 2-3 weeks in which the students focus on understanding the requirements and transforming the problem statement into a prioritized product backlog. In this so-called *Sprint 0*, we encourage students to create initial informal models to communicate their ideas concerning the problem domain and the system architecture to their fellow team members and the client [7]. This allows the team to establish a shared mental model of the system they will develop and creates a mutual understanding of possible technologies involved. The project teams typically create a Top-Level Design as well as a Analysis Object Model during this Sprint 0. Most teams choose to use Gliffy[3] as a modeling tool because it is well-integrated into the existing knowledge management tools used during the course and offers a mix of semi-formal and informal templates. Together with the modeling cross-project team members, we review these two models in a feedback meeting, in which one representative from each team presents their models and the corresponding high-level architectural decisions. In subsequent sprints the teams continuously refine these models every time new requirements arise or existing requirements change.

We introduce the remaining two perspectives of our modeling approach as well as techniques on how to present the models in a course-wide lecture held four weeks into the course. As their project becomes more concrete in terms of system components that are needed to fulfil the required functionality, the teams refine their two initial models and iteratively develop their Subsystem Decomposition. When they make decisions concerning concrete technologies, protocols, and hardware used, they create their Hardware-Software Mapping, which completes the four-perspective approach.

[2]https://youtu.be/sNVMtK8CcbM

[3]https://www.gliffy.com

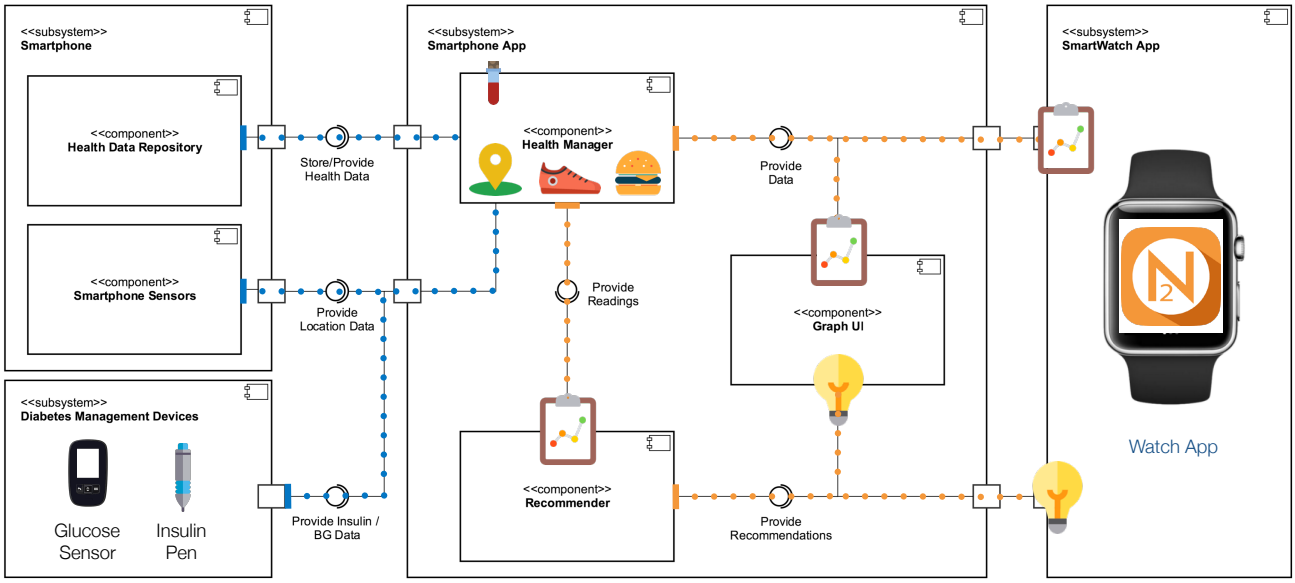

Fig. 3: Subsystem Decomposition containing icons for the visualization of data flow.

After further peer-to-peer feedback rounds with their project leader and experienced students who are familiar with our approach, each team presents their relevant models in a course-wide presentation which we call *Design Review* two months into the course. For this presentation, we teach the students to tell a story along with the models and to transform each model from the four-perspective approach into one another. For instance, the students present the relevant objects of the problem domain and their relations in the Analysis Object Model, and animate the same objects through their Subsystem Decomposition diagram to show the data flow. While preparing their presentation, students switch from modeling tools to presentation software such as Apple Keynote[4] to integrate high-resolution icons and transitions. Keynote allows for a seamless integration with other content, such as requirements descriptions or status updates. We provide an example of presenting the above-mentioned models as an online video.[5]

At the end of the course we ask each team to document their results using the four models created. For each model they create navigational text that describes the key elements and their relationships. This document in combination with the other artifacts created during the course, such as the actual source code repositories, setup instructions for the infrastructure, or videos of the teams' presentations, are delivered to the clients. The delivery marks the end of the course.

V. Discussion

We have been teaching the four-perspective modeling approach in our project courses for over 5 years during which it has been applied in over 100 student projects. In this section, we reflect on the experiences we made.

[4]https://www.apple.com/keynote/
[5]https://youtu.be/sNVMtK8CcbM

Firstly, using a modeling approach that diverges from known standards most students are already familiar with (such as UML) certainly involves additional teaching effort from the instructors' side, as well as added learning effort for the students. We go through several feedback cycles before the Design Review to support the teams in preparing their models. This is perceived as additional workload by students which they do not always see as necessary, especially in the first month of the course. As they begin using the models for communication with their client, they start to see the usefulness of the modeling approach for communicating their system architecture. Students' perception improves even further as the Design Review approaches and they start preparing their presentations. While we allow the students to decide which models to present, most teams choose to show three or even all four models in their presentation, transitioning from each perspective to show a different view of their system.

Our experience shows that clients welcome the graphical models and they use the Analysis Object Model to arrive at a shared understanding of the problem domain early in the project. Seeing how the students understood the problem gives them the opportunity to correct assumptions or clarify questions with their extensive domain knowledge. The Top-Level Design and Subsystem Decomposition are used to discuss architectural decisions. The Hardware-Software mapping is used internally by the team to discuss the deployment of the components, but rarely shown to the customer due to its comparably high level of technical detail.

One may argue that the applied modeling techniques do not conform to the requirements of (semi-)formal modeling. While we provide a basic set of notations for each model, we encourage the students to integrate creative ideas that help to understand the models' content and to make adjustments to the models' syntax during discussions with customers.

While some models do break syntactic rules of UML, we think that they make up for it in understandability and their suitability to express the requirements of a wide range of projects. By removing strict modeling rules, we reduce initial hurdles faced by clients with a non-technical background and thus allow them to be involved in discussions about system design decisions. Furthermore, the use of informal models encourages involved discussion partners to focus on the relevant parts of the system rather than syntactic details of the model [7]. We also encourage students to strictly distinguish between technology-independent and platform-specific representations to focus on the problem domain rather than getting lost in discussion about the solution domain. The results of each perspective can subsequently serve as a basis for additional iterations that fully comply with UML specifications.

The anecdotal evidence we gathered during the past years indicates that while modeling is perceived as additional effort at the beginning of the course, both students and clients see the advantages of the approach and rely on it in their presentations and their regular communication.

VI. Conclusion and Future Work

In this paper, we described a four-perspective modeling approach to overcome the challenge of enabling audiences with limited technical knowledge to understand the architecture and inner workings of a software system. The approach consists of a Top-Level Design, Analysis Object Model, Subsystem Decomposition and Hardware-Software Mapping.

It emphasizes the fast creation of models towards a commonly shared understanding of the system at hand. In addition, we outlined a teaching method for iteratively creating the model for each perspective which includes multiple roles and responsibilities, feedback rounds and presentations. In the last five years, we used the four-perspective modeling approach in over 100 student projects with industry partners in the context of a large capstone course. The feedback we received from students, project leaders and clients as well as the successful implementation of the projects validate the usefulness of the four-perspective modeling approach on an anecdotal level.

In the future, we plan to further structure and evaluate how development teams work with the four-perspective modeling approach. In particular, we want to identify which aspects help the most to transport key elements of a software system. Another interest of ours is the influence of modeling on learning outcomes in terms of an enhanced understanding of software architecture and technologies.

References

[1] L. Alperowitz, C. Scheuermann, and N. von Frankenberg. From storyboards to code: Visual product backlogs in agile project courses. In *Tagungsband des 15. Workshops "Software Engineering im Unterricht der Hochschulen" 2017, Hannover, Deutschland*, pages 69–72, 2017.

[2] C. Bastarrica, D. Perovich, and M. M. Samary. What can Students Get from a Software Engineering Capstone Course? *International Conference on Software Engineering*, 2017.

[3] B. Berenbach, F. Schneider, and H. Naughton. The use of a requirements modeling language for industrial applications. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 285–290. IEEE, Sept 2012.

[4] B. Bruegge and A. H. Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 2010.

[5] B. Bruegge, S. Krusche, and L. Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education (TOCE)*, 15(4):17, 2015.

[6] R. Chatley and T. Field. Lean Learning - Applying Lean Techniques to Improve Software Engineering Education. *39th International Conference on Software Engineering*, 2017.

[7] D. Dzvonyar, S. Krusche, and L. Alperowitz. Real projects with informal models. In *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014.*, pages 39–45, 2014.

[8] R. Lemma, M. Lanza, and F. Olivero. CEL: Modeling everywhere. In *International Conference on Software Engineering '13*, pages 1323–1326, USA, 2013. IEEE Press.

[9] M. Paasivaara, J. Vanhanen, V. T. Heikkil, C. Lassenius, J. Itkonen, and E. Laukkanen. Do High and Low Performing Student Teams Use Scrum Differently in Capstone Projects ? *International Conference on Software Engineering*, 2017.

[10] F. Schneider. *URML: Towards Visual Negotiation of Complex System Requirements*. Dissertation, Technische Universität München, München, 2016.

[11] D. Wüest, N. Seyff, and M. Glinz. FlexiSketch team: Collaborative sketching and notation creation on the fly. In *International Conference on Software Engineering '15*, pages 685–688, USA, 2015. IEEE Press.