A Game for Taking Requirements Engineering More Seriously

Eric Knauss, Kurt Schneider *and* Kai Stapel
Software Engineering Group, Leibniz Universität Hannover
Welfengarten 1, 30167 Hanover, Germany
{Eric.Knauss,Kurt.Schneider,Kai.Stapel}@inf.uni-hannover.de

Abstract

Requirements engineering (RE) is often neglected and not taken seriously. In particular, students and small or medium enterprises do not see how RE activities are related to the success or failure of projects.

We address this serious problem with a game students can play in a few minutes. Our software quantum metaphor helps to visualize a main challenge of RE: building the right system within available time. We animate the metaphor and present it as simulated software project. Agile ideas and comprehensive documentation can be tried out, as well as the impact of prototypes and reviews on requirements.

A player needs to balance speed and quality, and should weed out early misunderstandings later. This helps to experience advantages of systematic RE. The game facilitates comprehension – and encourages taking RE a little more seriously.

1. Introduction

Requirements engineering does not produce code immediately. Instead, interviews and workshops often uncover conflicting requirements. Resolving and meeting those requirements is difficult, and the value added is difficult to see.

Many small projects seem to get along without any systematic requirements practices. Requirements have been neglected for years, and there still seem to be successful projects. How could systematic requirements engineering improve the situation? Although these perceptions may be wrong or incomplete, this view is still wide-spread in industry [7].

As CMM and its successors have found, "heroes" (exceptionally good developers) may compensate for some poor requirement practices [9]. However, relying on those few heroes has shown to be a trap.

Unfortunately, students are almost encouraged to adopt the above-mentioned view on requirements. During their computer science education they acquire a lot of technical knowledge (e.g. algorithms and data structures, databases, artificial intelligence). In problem assignments they are rewarded for good implementations. When students adopt a programming job in a small company, they encounter that same attitude again: Value seems to reside close to code only. Requirements engineering is often perceived as a burden and a "soft issue" with little impact on project success. Uncovering requirements seems to add to the problem rather than to the solution. When students graduate from Universities, they are hired by a company and tend to reinforce the vicious circle.

In University education, there is only a limited range of opportunities for stimulating critical thinking. Limited time and limited size of work assignments in software engineering also delimit the insights we can convey. Of course, student projects are among the most important contribution to gather realistic experience.

We were searching for *additional* means to make students appreciate requirements engineering more. In particular, we intended to provoke their own thoughts instead of telling second-hand lessons. A small set of basic insights should be experienced in a self-directed way of learning.

For that purpose, we decided to build a web-based simulation game. It should offer a simple model that would be able to convey those insights. Unlike more ambitious approaches of simulation software projects [4, 10], our game should be easy to understand and fun to play within just a few minutes. Design and presentation of that game needs to attract students and practitioners. Its basic underlying concepts must be explained in only a few sentences; the core of the concept must be visualized. We do not attempt to cover all aspects of software project dynamics [1], but we focus on the tension between developing a system right, and developing the right system.

In section 2, we present related work. Section 3 is devoted to the goals of our game. Its conceptual foundation is described in section 4. This underlying model is visualized during the game in order to convey basic requirements engineering lessons. Section 5 presents our simulation-based game which was implemented during a one-semester project by a team of Bachelor students.

Third International Workshop on Multimedia and Enjoyable Requirements Engineering (MERE 08), Barcelona, Spain, 2008

2. Related Work

Requirements must be elicited and communicated within a software project. During interviews and workshops, customer requirements must be identified, stored, and reconciled with those of other stakeholders. In the next step, a designer or architect will derive decisions and other information related to the requirements. After several steps, one or more programmers will implement the decisions that were built on the requirements.

Several researchers have investigated dependencies and dynamics of software projects including the impact of requirements flowing through projects:

Abdel-Hamid and Madnick [1] developed quantitative simulation models of software projects. They were mostly interested in "policies" which they denote using the System Dynamics simulation concept [5]. For example, the number of *features-implemented* may depend on the number of *requirements-elicited*. Unlike prescriptive models of projects (like many software process models), their descriptive models allow decision makers to compare different alternative „policies" and study their impact on key project values such as *features-delivered, delivered-errors* etc.

Rodriguez et al. followed the System Dynamic approach of simulation [10] to support decision making.

SESAM is an interactive educational game based on a sophisticated model [11]. SESAM emphasizes the activities of a player („project leader"). It derives consequences in terms of project events during a simulated software project [4]. Using SESAM requires substantial amounts of time and preparation [4]. Even *playing* the game is intended to take some time. SESAM is a sophisticated educational tool.

In contrast, our web-based game reuses basic concepts of SESAM models but pursues different goals: Playing the game is supposed to be fun; students should be attracted by the desire to play successfully and to compete with their peers for a better score. Insights are conveyed as a by-product of playing, by watching the visualized flow of requirements.

3. Motivation and Goals

Although our game should be fun to use, it is supposed to visualize and convey a (limited) number of serious insights. As outlined above, those insights are not directly *visible* during *real* software projects. In particular, students should see and experience:

- Requirements and derived information (design, rationale) needs to flow from customers to code. This flow needs to reach or involve all stakeholders and project participants.
- This information can be stored in documents or in human brains for a while. Documents and brains have different characteristics. For example, documents take longer to fill and to retrieve from, while humans tend to forget.
- There is more than one possible path from customers to code. Some paths rely on documents (e.g., when following the V-model [6]), while others avoid documents (such as agile approaches [2, 3]). In most real projects, there will be a mixture of document-based an informal communication.
- When people never validate the information they receive, there will be misunderstandings and loss of original requirements.
- Software quality is defined with respect to project priorities: in a small and urgent project, covering many requirements fast will be a top priority. Other projects may prefer consistent documentation. Specifications and design documents that reflect customer requirements increase maintainability.
- There is a random element in software projects. A requirement can always be forgotten or lost, and one never knows exactly how much information will flow during a meeting.

We envisioned a light-weight game that should be fun to play in a few spare minutes. Our goals corresponding to this kind of game were:

- Students and interested practitioners should find the game on our web-site and be attracted to try their skills.
- Explanations should be limited to a minimum.
- Playing one game should take less than 10 minutes.
- There should be a few explanations during the game to help players reflecting and understanding the deeper meaning of what they see [13]. This is considered crucial for lasting insights.
- The game should encourage people to play it more than once, thus repeating and internalizing [8] the concepts and insights it conveys during interactive learning.
- The game should look nice and appealing, not like teaching material.

4. Basic Concepts: Software Quantums

Software quantums are the basic concept behind our simulation game. The "Software Quantum Metaphor" was first introduced in SESAM [8]. The name "Software Quantum" alludes to atomic units. One unit

of project information (e.g., requirements) is called a "software quantum" [4].

In SESAM, software quantums were used as an internal model. Software quantums were not shown or visualized in any way. In the game presented in section 5, visualizing software quantums is an essential novelty.

Requirements are visualized as a bag of balls. Each ball visualizes one software quantum (see Fig. 1). Customer requirements are all *postulated* to be legitimate and valid at this point. They are displayed in yellow (printed in light gray). When analysts interview a customer, they "grab a handful" of those balls and put them in their own "mental bag". In this bag, the incoming balls are added to the set of already present balls. Balls can be grabbed during each simulation step. However, some (false) requirements are also added. They do not represent customer requirements but misunderstandings or wrong assumptions (blue balls, printed as dark gray). False and invalid customer requirements are also modelled using this mechanism. The longer analysts interact with customers, the more balls they collect.

Mathematically, flow between bags is an operation of random selection from a set (with putting back) of n balls per time unit. Balls are individuals, so the same ball will not be added again to the analyst's mental bag. More and more different balls (right and wrong) will assemble over time. Therefore, chances to grab a new software quantum are decreasing. At the same time, additional gold-plating or misunderstanding (dark balls) keep being added. Requirements must be elicited long enough, but not too long.

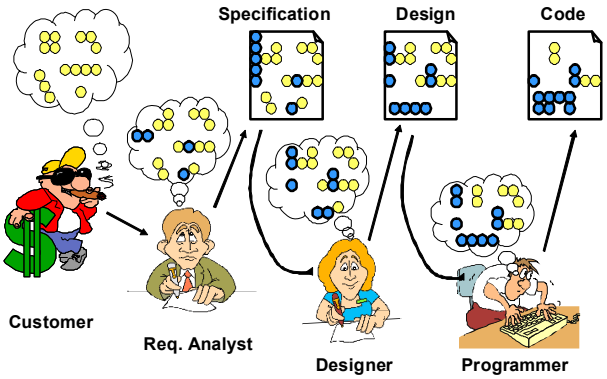

Figure 1: Flow of software quantums from customer through people and documents

Obviously, software quantum models can follow different processes, use different sets of documents and stakeholders, and contain many additional complications (e.g., changing customer requirements). For the sake of clarity, we refrain to the basic mechanisms.

A number of additional **axioms** make models useful with regard to the goals outlined above

- Each software quantum is an individual, represented by the unique position of a ball within alls document and mental bags.
- Software quantums are anonymous. All yellow balls together model all valid requirements in the project. It is unknown (and irrelevant for our purpose!) what a particular software quantum stands for. It just represents "a small part of all requirements".
- Moving one software quantum from a mental bag to a document (or vice versa) takes the same amount of time and effort, independently of its colour. Documents grow, but this may be due to "wrong requirements", too. Polluted flows of requirements are considered a main reason for building the wrong system.
- Colours of quantums are invisible to the simulated project participants. However, players can see them. This is essential for gaining insights.
- Reviews are an opportunity to compare sets of software quantums, and get rid of wrong ones.
- Requirements are right or wrong (light or dark) only with respect to current customer requirements. This definition accommodates several dynamic aspects like the impact of changing customer requirements.

5. A Game for Learning

The game presented in this chapter animates the software quantum metaphor. Flow rates ("number of balls grabbed") were calibrated to produce plausible effects, but there was no quantitative validation carried out. The purpose of the game is to convey the above-listed phenomena and insights – it is not intended for planning real projects.

A team of five Bachelor students of Computer Science at Leibniz Universität Hannover (LUH) implemented the simulator within one semester. Their task included design of interface appearance as well as software design and implementation as a Java applet. Applets can be easily integrated into websites and played remotely. There is no installation necessary. The game applet is accessible (in English) at www.se.uni-hannover.de/en/qgame.

During several sessions with the authors of this paper, students developed concepts to meet the goals and constraints. Immediately after completion, the game was presented in the lecture hall and released on the website. During the first period of use two severe and a number of less critical errors were found and removed.

5.1 Three-Level Game Design

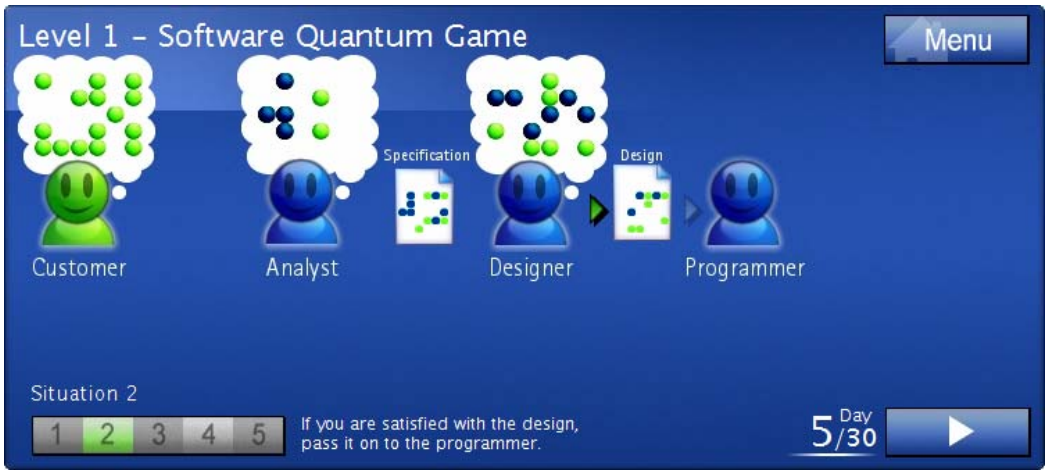

Fig. 2: Players choose flows and advance simulation time on level 1

When a player visits the site, the game offers three levels and a short (approx. half page) explanatory text. Players are advised to master one level after the other, like in other games.

Each level conveys a set of related concepts:

The **first level** (Fig. 2) familiarizes players with basic operations (letting software quantums flow from a source to a target; advancing simulation time; going to the result page). On this level, software quantums can only flow along a pre-designed path. It resembles Fig. 1 and follows a traditional waterfall process.

Level 2 extends level 1 by adding alternative flows. Players can not only choose *when* to activate the "next" flow, they can also decide if this flow should use documented (as in level 1) or oral communication between people. In the latter case, direct arrows between mental bags can be activated instead of flows to and from documents.

Level 3 adds more variants. For example, customers can choose to talk to programmers directly (Fig. 3). This helps to avoid intermediate steps and delays. At the same time, all specification or design documents are skipped. Such a project is less maintainable and depends on individual customers much more – and suffers when they change their minds.

Level 3 also introduces the option of scheduling reviews (Fig. 3). Two documents are selected by the player. A review compares the given documents or bags and removes some of the differences. This operation costs simulated time. When it is carried out too often, there will not be sufficient time left for making requirements flow. Without reviews, however, documents get polluted with false requirements.

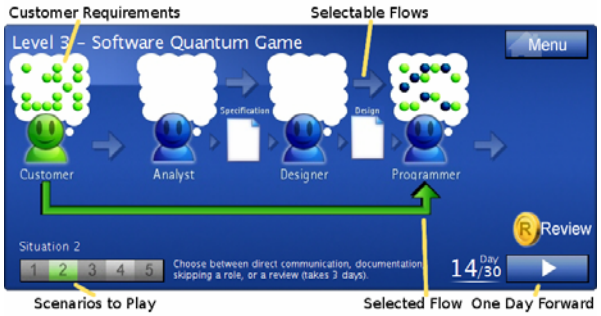

Fig. 3: More variants for flows and reviews

5.2 Results and Reflection

Each project lasts for 30 simulated days. Players decide which requirement flows they want to activate by selecting one arrow at a time (highlighted in Fig. 3). Then they push the "forward" button to advance the simulation by one day. After 30 days have expired, players meet their simulated "boss" (Fig. 4). Depending on a comparison of customer requirements (left) and delivered code document (centre), the reaction of the boss (right) varies.

Fig. 4: Feedback of results – facts and emotions

Fig. 4 shows a negative ("Disasterous") and a positive overall reaction ("Well done"). Statistics (left) support that reaction and provides a score for competition. The first priority in evaluating results is the percentage of correct customer requirements delivered. The number of false software quantums indicates misunderstandings, unwanted features and wasted effort. They are, therefore, deducted from the score. This simple rule is explained to players in text.

The text coming with this evaluation explains both the rules of the game and the rules of software projects: Implementing unwanted features is detrimental to success, while as many correct requirements as possible should be implemented.

By choosing between oral (more precisely: *fluid* [12]) and document-based communication in levels 2 and 3, players are made aware of those alternatives – they exist in real projects, too.

According to the above-stated goals, players can decide on flows, but there is a random element determining how many (dark and light) quantums flow per simulated day. Five different situations (i.e., seeds for the random generator) can be chosen on each level. As Fig. 2-4 show, the same seed (selected situation "2" in all those Figures) leads to precisely the same pattern of customer requirements.

This enables players to repeat, improve, and compete under identical circumstances.

6. Conclusions

Requirements engineering is an important and serious activity within a software project. Nevertheless, it is often neglected due to missing awareness for RE options and impact of those options.

In our fast and simple game, we want to convey a few very basic and essential insights. We reused the software quantum metaphor, decided to visualize it as colored balls, and presented it as a project simulation game. The main challenge lies in avoiding wrong requirements and collecting right ones. Reviews help in this challenge, but they cost time needed elsewhere for pushing requirements forward.

A "boss" evaluates and comments on results. Short texts lead from one game level to the next. By learning about new options in the game, students also learn and experience important options and phenomena in software projects: The importance of oral communication and reviews, alternative routes of requirements flow, and the impact of different quality requirements are playfully introduced.

Practitioners and even customers in software projects are invited to play and use the game as a stimulus for discussing flow of requirements in their own project, as well as their own quality priorities.

This game will not solve all problems of requirements engineering. However, it can contribute to taking requirements engineering a little more seriously - while having fun!

8. References

1. Abdel-Hamid, T. and S.E. Madnick, *Software Project Dynamics*. 1991, Englewood Cliffs, NJ: Prentice Hall.
2. Beck, K., *Extreme Programming Explained*. 2000: Addison-Wesley.
3. Beedle, M. and K. Schwaber, *Agile Software Development with Scrum*. 2001: Prentice Hall.
4. Deininger, M. and K. Schneider. *Teaching Software Project Management by Simulation - Experiences with a Comprehensive Model. Conference on Software Engineering Education (CSEE)*. 1994. Austin, Texas.
5. Forrester, J.W., *Industrial dynamics, 9. Edition*. 1977: Cambridge.
6. IABG. *V Model, www.v-model.iabg.de*
7. Knauss, E., et al., *Nicht perfekt, aber richtig - Erfahrungen mit Software-Anforderungen.* Softwaretechnik Trends, 2007. 27(1).
8. Nonaka, I. and H. Takeuchi, *The Knowledge-Creating Company*. 17 ed. 1995: Oxford University Press.
9. Paulk, M.C., et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*. 1 ed. SEI Series in Software Engineering, ed. M.C. Paulk, et al. Vol. 1. 1994, Reading, MA: Addison Wesley Longman, Inc..
10. Rodriguez, D., M. Satpathy, and D. Pfahl. *Effective software project management education through simulation models. An externally replicated experiment. Conf. on Product Focused Software Process Improvement (PROFES)*. 2004. Kansai Science City, Japan: Bomarius, F.
11. Schneider, K. *A Descriptive Model of Software Development to Guide Process Improvement. Conquest*. 2004. Nürnberg, Germany: ASQF.
12. Schneider, K. *Generating Fast Feedback in Requirements Elicitation*. Conf. on *Requirements Engineering: Foundation for Software Quality (REFSQ 2007). Trondheim, Norway*.
13. Schneider, K. and K. Nakakoji. *Collaborative Learning as Interplay between Simulation Model Builder and Player. Computer Supported Cooperative Learning (CSCL)*. 1995. Indianapolis: IN.