Using Software Theater for the Demonstration of Innovative Ubiquitous Applications

Han Xu
Technische Universität München
Boltzmannstr. 3, Garching, Germany
han.xu@tum.de

Stephan Krusche
Technische Universität München
Boltzmannstr. 3, Garching, Germany
krusche@in.tum.de

Bernd Bruegge
Technische Universität München
Boltzmannstr. 3, Garching, Germany
bruegge@in.tum.de

ABSTRACT
Software development has to cope with uncertainties and changing requirements that constantly arise in the development process. Agile methods address this challenge by adopting an incremental development process and delivering working software frequently. However, current validation techniques used in sprint reviews are not sufficient for emerging applications based on ubiquitous technologies. To fill this gap, we propose a new way of demonstration called Software Theater. Based on ideas from theater plays, it aims at presenting scenario-based demonstration in a theatrical way to highlight new features, new user experience and new technical architecture in an integrated performance. We have used Software Theater in more than twenty projects and the result is overall positive.

Categories and Subject Descriptors
D.2.1 **[Requirements/Specification]**: Elicitation methods D.2.2 **[Design Tools and Techniques]**: Evolutionary prototyping

General Terms
Design, Human Factors

Keywords
Informal models; Prototypes; Scenarios; Rapid iteration; Demonstration; Design evaluation

1. INTRODUCTION
Requirements engineering is a creative process [1] that is filled with uncertainties and changes. Dealing with uncertain and changing requirements is particularly challenging in terms of stakeholder communication and design validation [2]. Agile methods address this challenge by adopting an incremental development process and delivering product increments frequently [3]. Prototyping is being used to evaluate design ideas quickly. By focusing on only important aspects of the system and ignoring irrelevant details, prototyping allows us to get feedback from the stakeholders without having to fully implement the system. As prototypes alone do not provide enough context of the usage, scenarios can be used as a complement [4]. However, this is still not sufficient when it comes to exploratory projects based on emerging technologies (e.g. ubiquitous computing). These projects are developing new products in the market and have to deal with uncertainties coming from both the application domain and the solution domain [5, p.41]. They require the exploration of new features, new user experience and new technical architecture as a combination, which we call *integrated new design*. In order to provide an efficient and reliable evaluation of this *integrated new design* with the stakeholders before entering the product implementation process, we propose *Software Theater*, a new way of demonstration. Software Theater borrows ideas from the theater play, aiming to present scenario-based demonstration in a theatrical way to highlight new features, new user experience and new technologies as a whole. We have used Software Theater in more than twenty projects ranging from wearable computing, Internet of Things (IoT) to mobile applications and the result is overall positive. Software Theater is reported as stimulating insightful feedback and receiving more positive confirmation about the design from stakeholders.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08...
http://dx.doi.org/10.1145/2786805.2803207

2. DEMONSTRATION AND EVALUATION
2.1 Demo Using Prototypes and Scenarios
One of the major activities of stakeholder involvement is the evaluation of design. This is achieved by demonstrating the functional and non-functional aspects of the current design to the stakeholders and expecting feedback from them. The demonstration can be conducted using prototypes of different fidelity levels depending on which is appropriate in the given situation. Ideally the design should be demonstrated and evaluated when there is a change that may cause significant consequences. Compared to fully implemented systems, prototypes allow us to evaluate design ideas more quickly and at a lower cost. This is achieved by defining appropriate focus and choosing the right form of prototype for the given situation [6, p.115].

As prototypes alone do not provide enough context of the usage, they are often used in combination with scenarios [4]. Scenarios, as concrete description of the system usage, provide a bridge between the *usage world* and the *system world* [7] and are helpful in making sound design decisions by focusing on both the problem space and the solution space [7][8]. Scenarios are intuitive and suitable for communication and validation. As Rolland et al. stated, "People react to 'real things' and ... this helps in clarifying requirements." [9] The story-like description with context information makes it easier for stakeholders to understand abstract concepts in the system design. Scenarios are cost-efficient and enable quick iterations. In a changing environment (as is nearly always the case in software development), the design of the system often takes several revisions to reach a "stable" state. Therefore, it could be costly if an executable system were developed in every iteration. Instead,

scenarios are cheaper to create and modify; they provide an ideal compromise between cost and efficiency, especially in the early iterations of innovative projects. Apart from the economic factor, scenarios are open-ended and stimulate the user's imagination. They enable the users to come up with more specific requirements and help "the analysts to consider contingencies they might otherwise overlook" [10]. There are different ways to use prototypes with scenarios depending on who actually do the demonstration. It can be either *user-performed*, where users are provided with scenarios as description of task and are told how to use the prototype to perform the task [11, p.459], or *developer-performed*, where scenarios provide a context in which the prototype is demonstrated how to achieve specific tasks [12]. As Weidenhaupt et al. reported [4], combining the development of scenarios and prototypes enable stakeholders to check, discuss and update scenarios and prototypes at the ground level, and provides better customer satisfaction.

2.2 Problems for Ubiquitous Applications

As little can be learned from the past, innovative applications based on Internet-of-Things (IoT) and wearable computing technologies are faced with more challenges in designing features, user experience and system architecture. As Jarke et al. described: "in these innovation-driven settings, requirements become part of both the business solution and the system solution, and they constantly bridge new solutions to organizational and societal problems … revisiting requirements as implementation progresses and emphasizes the dynamics and intertwining of these activities" [13]. These applications are new in the market and often come with new features, new user experience and new technical architecture as a combination, which we call *integrated new design*. In order to perform a cost-efficient but reliable demonstration for these applications, we need to solve the following problems:

- (P1) We need a prototype that can embody the *integrated new design* (most likely a combination of new features, new user experience and new technical architecture).
- (P2) We need a way to present this prototype in a specific context to evaluate the applicability of the feature, the usability of the user interface and the feasibility of the architecture (e.g. the performance of the sensors and devices).
- (P3) Unlike desktop or ordinary mobile applications, ubiquitous applications are supposed to react to or interact with the environment. Thus the demonstration cannot work out without the participation of the environment.

3. SOFTWARE THEATER

3.1 The Benefits of Theatrical Techniques

Combining prototypes and scenarios has been proved to be a useful way to enhance design and user participation in the demonstration session [4][12][14]. In a step further, theater plays can be used as a way to present software design, which have the benefits inherited from scenarios and role-playing [10][15][16], and have been used in requirements elicitation and usability studies [15][17]. The benefits of theatrical techniques can be summarized as:

- Increase mutual understanding among stakeholders
- Stimulate imagination of team members
- Leave rooms for opened-ended improvisational performance
- Arose the empathy of the actors and the audience
- Highlight existing problems and benefits of the new design

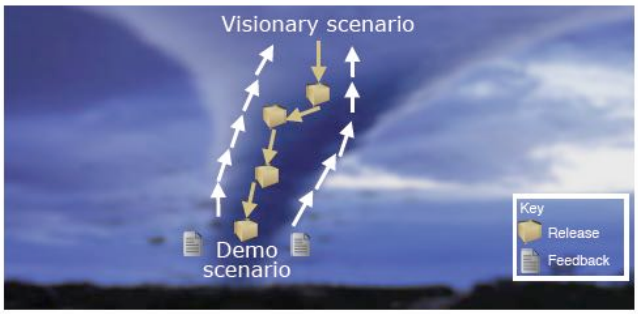

Figure 1. Tornado model: Wide in analysis, narrow in implementation [18]

3.2 What is Software Theater?

In response to P2 mentioned in Section 2.2, Software Theater is a way to perform software demonstration in a theatrical way so that stakeholders can evaluate the applicability of the feature, the usability of the user interface and the feasibility of the technical architecture. Traditionally, prototypes are presented in a non-engaging way, generating only limited empathy with the demonstrated system. However, presentations without a lifelike context are too "dry" for the demonstration of innovative applications based on Internet-of-Things (IoT) and wearable computing technologies, because these applications are unseen in the market before, it is hard for people to understand the purpose and usage of the new application by just looking at the user interface, etc. Software Theater, instead, creates a vivid atmosphere, which, through the performance of the actors, highlights how to use the new application to solve existing problems and make everyday life easier. Software Theater can be used with both partially-implemented prototypes and fully-implemented systems depending on the stage the demonstration is used, which will be discussed below.

3.3 Demo-Oriented Development Using Tornado Model

In response to P1, when Software Theater is used with a prototype in the middle of a project (Design Review in our case), we need an executable demo system that "just fits", representing the *integrated new design* under evaluation (features, user experience, and technical architecture) "no more no less". The basic tenet is to use higher fidelity prototypes for the relevant parts under evaluation and lower fidelity prototypes for irrelevant parts. In order to support creation of this demo system, we propose using Tornado Model [18][19].

The Tornado model is a demo-oriented development process aiming to deliver "touchpoints" (see Figure 1), a metaphor for creating executable prototypes in order to evaluate design ideas and obtain feedback from the stakeholder. The Tornado model stresses the role of informal models in closing the gap between the *design model*[1] and the *user model* [18][20]. Informal models, as a means for communicating with stakeholders, focus on the look-and-feel and user interaction of the system. Examples of informal models include sketches, paper prototyping, low-fidelity user interfaces, storyboards, text-based scenarios and video-based

[1] Note that the *design model* used here is a mental model of the designer and should not be confused with the "design model" in object-oriented design.

Figure 2. Evolution of the user interface, from rough sketch (left) to the delivered application (right) ([21])

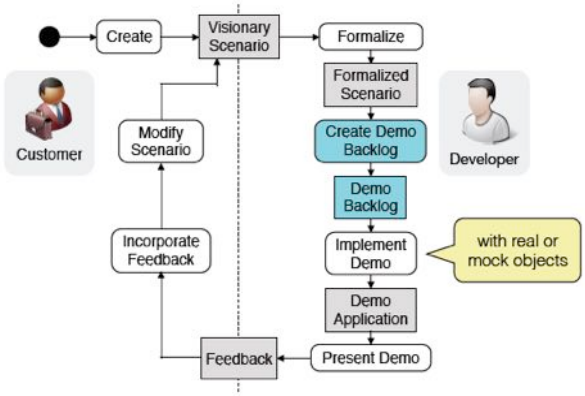

Figure 3. Software Theater activities

scenarios [22]. In this sense, Software Theater is yet another type of informal model.

The Tornado model employs different kinds of evaluation techniques at different stages of software development. Figure 1 shows a process that starts with visionary scenarios and funnels down to demo scenarios. *Visionary scenarios* represent the design ideas of the future system and are used for requirements brainstorming. In practice, they often require several rounds of iterations to reach a stable version. As the main task at this stage is exploring the problem space, low-fidelity prototypes are sufficient; visionary scenarios are created using textual description. *Demo scenarios* are refinements of visionary scenarios for reviews and presentations. They provide a demonstration of how the problem is addressed when using the system and can be played out in a demo. Demo scenarios are based on a working (or partially working) system, and often take advantage of mockups for cost-efficiency reasons.

The Tornado process is an evolutionary scenario-based design process. The initial version of the design is depicted using low-fidelity prototypes (for example, a sketchy user interface created on paper, see Figure 2, left). Low-fidelity prototypes are used in the early stages in an effort to get user feedback about the user interaction design as early as possible. This enables the user to explore possible design alternatives and reformulate the initial requirements. In the middle of the project, as only promising alternatives are left, interactive prototypes (for example, software mockups created with Balsamiq as shown in Figure 2, middle) are used for a more tangible and reliable evaluation of the requirements, user experience and system design. At the end of the project, the finally adopted design is implemented and delivered (as shown in Figure 2, right). A tornado is wide in the clouds, but only a part of it funnels down and hits the ground at its *touchpoint*. The touchpoint is where an executable demo system is created and presented. It is by this metaphor that we give it the name Tornado Model.

3.4 The Workflow of Software Theater

Similar to performing a prototype based on predefined scenarios, Software Theater is performed based on a screenplay. The screenplay describes the event flow of the demo, the cast (that is, the participating actors), and the props required for the demo. The purpose of Software Theater is to demonstrate how end users would benefits from the new product in the real world context.

We take the following workflow to create the demo, prepare the screenplay and perform the demonstration (see Figure 3). The first activity is that the team identifies visionary scenarios to be demonstrated and then turn them into formalized scenarios (if not yet exist). A formalized scenario describes the same content as the visionary scenario, but in a structural way. "Formalization helps to identify areas of ambiguity as well as inconsistencies and omissions in a requirements specification" [5, p.174]. Next, the team creates the screenplay by deriving the event flow as well as the participating actors from the formalized scenario and identifying the props and stage directions needed for the performance. Then, they identify the subsystems and services that are required to realize the demo. While services that require technical evaluation (e.g. performance-critical or user experience-significant features) should be added in the demo backlog as action items for actual implementation, other services could be mocked for both environment simulation (as a response to P3 mentioned in Section 2.2) and cost-efficiency reasons. The demo backlog contains all the action items to realize the demo. When the demo is delivered, it is presented by the actors according to the screenplay. After the demonstration, feedback is collected and incorporated to update visionary scenarios and the design.

4. CASE STUDY

We regularly conduct a capstone course called iOS Praktikum, which takes up to 100 computer science students to develop innovative applications for industry partners in separate teams. Our multi-project organization, which was explained in detail in [18], permits several software engineering projects to run in parallel. These projects are often expected to create new features, new user experience or to use new technologies (such as wearable devices, smart home sensors etc.). In general, the main objective of these pilot projects is to develop an executable prototype proving the practicability of the application. Therefore, Software Theater was adopted on the one hand to evaluate the feasibility of the design and on another to communicate with the customers with different technical background. Software Theater was used in two major presentations: *Design Review* that takes place after two thirds of the project, and *Customer Acceptance Test* as the final presentation of the project [18]. On both occasions, the participants should perform a live demonstration of their applications using Software Theater. Recordings of these presentations are available on the project website [23]. The result of applying Software Theater technique in these projects is overall positive according to our preliminary stakeholder survey. In the following we share our findings regarding the technique:

- Software Theater strengthens the benefits inherited from scenarios by presenting scenarios using real people according to the screenplay in a lifelike scene – this puts the audience personally "on the scene" and gives them more empathy with the demonstration.

- Software Theater leads to insightful feedback on the new feature and new user experience.
- The screenplay is very important and has significant impact on the quality of the demonstration. When creating a screenplay, the team should take advantage of the theatrical nature of the demonstration and try to highlight the existing problems (e.g. the pain points of the user) and the benefits of the new application by using appropriate props and stage montage (such as projectors and audio effects).
- When we demonstrated prototypes without Software Theater in the past, customers seemed to hesitate to verify "I know this is what I want", even when it was actually the case; their feedback was more about falsification: "I am sure this is not what I want". Software Theater seems to make customers more comfortable to give positive verification about the design. However, this requires further study in the future.

5. RELATED WORK

Mahaux and Maiden proposed using Improvisional Theater to support team-based innovation in the requirements engineering process [1][15]. The commonality of Improvisional Theater and Software Theater is that they both employ the form of theater as an effort to improve stakeholder communication and increase mutual understanding. But they differ in several aspects. First, the purpose of Improvisional Theater is to generate creative ideas in the requirements engineering process, while the purpose of Software Theater is to demonstrate and evaluate design ideas for innovative software projects. Second, Improvisional Theater, as it name suggests, takes advantage of unplanned improvisional performance to stimulate the creativity of team members, while Software Theater emphasizes a predefined screenplay to set a framework for the demonstration. Third, Software Theater presents not only the applicability of user requirements, but also the feasibility of system requirements such as architecture design and hardware performance. To support this, Software Theater needs to be used in combination with specific software process and prototyping techniques (Tornado Model in our case).

6. CONCLUSION AND FUTURE WORK

In this paper we introduced Software Theater, a new way of demonstration for innovative applications based on emerging technologies such as wearable computers and Internet-of-Things (IoT). According to our experience of applying this technique in more than twenty projects, it is useful in evaluating new features, new user experience and new technical architecture (or *integrated new design*) that come as a combination with innovative ubiquitous applications. In future work, we plan to investigate different variations of performing Software Theater (e.g. allowing people outside the development team to be actors) and identify more guidelines to direct practice. We also want to conduct a more rigorous evaluation of this demonstration technique.

7. REFERENCES

[1] Mahaux, M. and Maiden, N.. 2008. Theater improvisers know the requirements game. *IEEE Software*.

[2] Xu, H., Creighton, O., Boulila, N., and Bruegge, B. 2012. From Pixels to Bytes: Evolutionary Scenario Based Design with Video. In *Proceedings of FSE 2012*.

[3] Beck, K., Beedle, M., Van Bennekum, A., et al. 2001. *Manifesto for Agile Software Development*. [Online] Available: http://agilemanifesto.org/.

[4] Weidenhaupt, K., Pohl, K., Jarke, M., et al. 1998. Scenarios in system development: current practice. *IEEE Software.*

[5] Bruegge, B. and Dutoit, A. H. 2010. *Object-Oriented Software Engineering: Using UML, Patterns, and Java (Third Edition)*. Prentice Hall.

[6] Arnowitz, J., Arent, M., and Berger, N. 2010. *Effective Prototyping for Software Makers*. Morgan Kaufmann.

[7] Jarke, M. and Pohl, K. 1993. Establishing Visions in Context: Towards a Model of Requirements Rrocesses. In *Proceedings of ICIS 1993*.

[8] Jarke, M., Klamma, R., Pohl, K., and Sikora, E. 2010. Requirements Engineering in Complex Domains. In *Graph Transformations and Model-Driven Engineering.* Springer.

[9] Rolland, C., Ben Achour, C., Cauvet, C., et al. 1998. A Proposal for a Scenario Classification Framework. *Requirements Engineering*.

[10] Carroll, J. M. 2000. *Making Use: Scenario-Based Design of Human-Computer Interactions.* The MIT Press.

[11] Pohl, K. 2010. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer.

[12] Sutcliffe, A. 1997. A Technique Combination Approach to Requirements Engineering. In *Proceedings of RE1997*.

[13] Jarke, M., Loucopoulos, P., et al. 2011. The Brave New World of Design Requirements. *Information Systems*.

[14] Sutcliffe, A. G. and Sawyer, P. 2013. Requirements Elicitation: Towards the Unknown Unknowns. In *Proceedings of RE2013*.

[15] Mahaux, M., Heymans, P., and Maiden, N. 2010. Making it all up: Getting in on the Act to Improvise Creative Requirements. In *Proceedings of RE2010*.

[16] Grudin, J. 2006. Why Personas Work: The Psychological Evidence. In *The Persona Lifecycle: Keeping People in Mind Throughout the Product Design*. Morgan Kaufmann.

[17] Newell, A. F., Carmichael, A., Morgan, M., and Dickinson, A. 2006. The use of theatre in requirements gathering and usability studies. *Interacting with Computers*.

[18] Bruegge, B., Krusche, S., and Wagner, M. 2012. Teaching Tornado: From Communication Models to Releases. In *Proceedings of the 8th edition of the Educators' Symposium*.

[19] Bruegge, B., Krusche, S., and Alperowitz, L. 2015. Software Engineering Project Courses with Industrial Clients *ACM Transactions on Computing Education*.

[20] Norman, D. 1996. Cognitive Engineering. In *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates Publishers.

[21] Dzvonyar, D., Krusche, S., and Alperowitz, L. 2014. Real Projects with Informal Models. In Proceedings of the 10th Educators' Symposium.

[22] Xu, H., Creighton, O., Boulila, N., and Bruegge, B. 2013. User Model and System Model: the Yin and Yang in User-Centered Software Development. In Onward! 2013.

[23] Krusche, S., Alperowitz, L. and Bruegge, B. *iOS Praktikum, Technical Universität München*. 2014. [Online]. Available: http://www1.in.tum.de/ios14.