Customer-oriented Development of Complex Distributed Systems

Ivonne Erfurth
Friedrich-Schiller-University Jena
07743 Jena, Germany
ivonne.erfurth@inf.uni-jena.de

Abstract

Complex and distributed systems are more and more common. Hardware is going from strength to strength and is embedded in high performance peer-to-peer networks mostly. The task of a software engineer is to develop software systems which are able to take part in these new possibilities. Hereby, the drawback is the simple fact that such software systems and their modeling are getting more and more complex.

If we are looking at the difficulties between customer and developer teams, especially misunderstandings between both, then the challenge to model customer oriented systems is even higher. For customers it is hard to understand the frequently used terms, process models, and technological concepts. Developers have a hard time to understand domain specific processes and structures, and exhibit a tendency to abstract concrete examples to higher level constructs. These problems are especially hard to avoid during the development of dynamic, distributed systems with multiple nodes and possibly asynchronous behavior.

In our research we develop a customer-friendly reference model to demonstrate the aspects of dynamic distributed systems understandable to the customer. This model presents and simulates the dynamic aspects of (distributed) systems without immediate abstraction from examples and allows for a stepwise generalization and evaluation with help of the customer team. In its final version the reference model will serve as a requirements statement for the professional developer.

There are two issues we are going to discuss before we introduce our approach. First, we shortly survey the situation in the distributed systems domain. Secondly we pinpoint the importance of customer centered modeling by reflecting on the usual success rate of software projects.

1 Modeling Distributed Systems

Modeling and designing software systems is a widespread and well researched job profile. The modeling of truly distributed systems is currently gaining interest and will be the dominating development profile in the future. However, its structure and work-flow have not been the focus of major research efforts so far. In this paper we will restrict our discussion to the distributed systems domain.

Most modeling approaches in this domain are based on Petri Nets. Already at the end of the eighties the first dedicated models for distributed systems were published. Krämer [6] combines Petri Nets and abstract data types into a formal, semi-graphical language, SEGRAS. Kindler [5] also uses Petri Nets to achieve a modular design in a distributed structure. See also Object Petri Nets [18]. Many others followed the same trade-off.

Petri Nets are abstract, formal models with strict rules for static specification and dynamic behavior. Because of this, Petri Nets are hard to understand for customers with little experience in computer science. Thus, they should be available to the developer, but strictly hidden from the customer.

Another approach emanates from the Unified Model Language (UML). Emmerich [1] represents UML-notations, e. g. use case diagrams, class diagrams for distributed object design. The major disadvantage is that UML has not yet fully formal semantics, at least in most of its notations. An online questionnaire [8], distributed and evaluated by our team, shows again that object oriented modeling is not easily understood by non computer scientists.

Giese [4] brings together both approaches. He combines object oriented modeling with Petri Nets in OCoN. However, the problem of possible misunderstandings between the customer and developer teams is still prevalent. In our opinion the logical conclusion must be to make sure that on the one hand the model of a possibly complex, distributed system is formal and provably consistent for the developer team and, on the other hand, fully transparent and under-

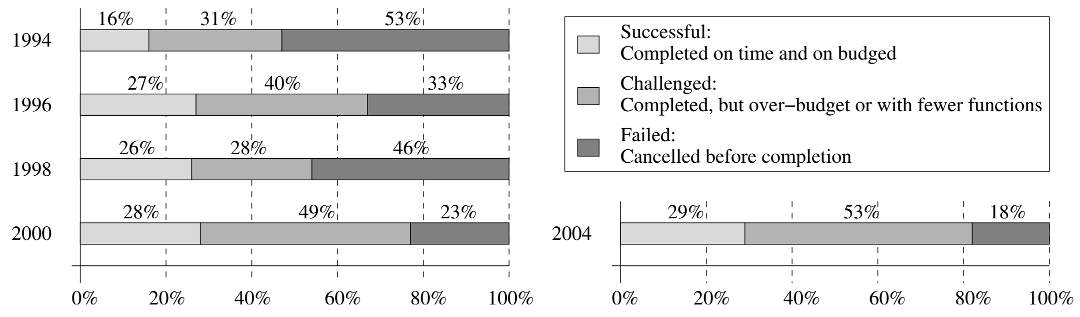

Figure 1. Success of software projects – Source [15, 16]

standable for the customer.

In the remaining portion of this paper we first discuss the well known problems of software development projects, with a special emphasis on the fact that in most all methodologies and tools the customer is not treated as a first class citizen. We believe this point to be one of the decisive issues in the development of distributed systems and will, therefore, offer a solution concept that integrates the customer via a separate reference model that features different views for different user groups. This will allow to integrate an informal and example-based view for the customer with a formal and type-based view for the developer. We conclude our presentation with a look at our working status and a discussion of future work.

2 Success of Software Projects

As most developers are no doubt aware, often the development of software is canceled in late phases or is retarded for months. At regular intervals the Standish Group [13] published a research report about the success of software projects. Since the first release 1994 [14], every time about 70 percent of all projects miss their goals (Figure 1).

A study [10] on the state of IT project management in the UK carried out by Oxford University in 2003 is alarming. This study reported that only 16 percent of IT projects were considered successful.

In connection with modeling of complex software systems not only technical problems cause projects to fail. Mostly abstract and formal models, which provide a concise and clean basis for the developer, are not easy to understand by the customer.

Our online questionnaire [8] handed out to (local) small and medium-sized businesses arrived at similar results. Goals of the questionnaire were to get answers regarding the state of art in practice. Which process models, methods and notations will be used? When, where and how arise problems between customer and developer teams during the development process? In addition it was important for us to get an answer how the customer is integrated in the development process. We got back 65 answered questionnaires from 54 companies mostly with less than 50 employees.

As one conclusion of the questionnaire we got a feedback about difficulties during the development process (see Figure 2). In addition to

(1) imprecise customer requirements and
(2) changes in requirements

also

(3) customers are not familiar with common techniques in software engineering and
(4) developers are not familiar with customer's domain problems.

This causes major problems during the software development process. The figure shows the percentage of participants who have indicated choices of the question: What do you think are reasons for difficulties during the software development process?

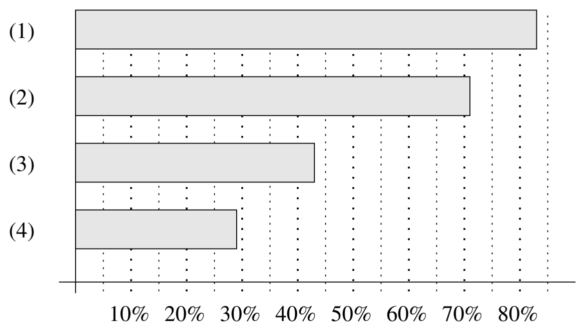

Figure 2. Reasons for deficiency in percent

Regarding the results of these studies we face two important points as success factors, as many others before:

1. User Involvement
2. Clear Statement of Requirements

Methods which involve users and support understandability, e.g. Participative Design [12] (STEPS [3], PICTIVE [7], C.A.R.D. [17]) are not widely spread. Not all, but most participative methods demand higher manpower requirements, skilled personnel and more time. This is involved with higher cost. Especially for small and medium-sized businesses this turns out to be too expensive.

To analyze the second factor – clear statement of requirements – investigations resulted in three main communication problems, mostly occurring in early phases of the software development process:

- Problems at the identification of requirements for systems to develop
- Misunderstandings because of different comprehension of terms
- Communication problems during the transformation of informal customer requirements into formal requirements for developers

Distributed and web-based applications, especially generalizations of the dynamics of such applications, are tough to comprehend for the customer and ask for well developed skills in abstraction and problem matching. This leads, in turn, to the effect that the customer either needs lots of attention to be guided, often on the basis of examples, through the abstracted, type-oriented model or simply disconnects from the development process and is later-on most likely disappointed with the results, causing project failure or costly re-development efforts.

At Friedrich-Schiller-University Jena (FSU) we suggest to develop a special *reference model* where the dynamics and distributed properties of a system are made visible, mostly on the level of concrete work-flow examples. We do not want to develop just another new notation or tool, which is not understandable to customers. We take an existing and often used notation as a basis. On a level above that notation we affix a customer-oriented and understandable additional notation. Between both notations is an automatism which translates the customer-oriented notation in the existing engineering notation. Derived abstract and formal models remain available to the developer but are hidden from the customer. In the sense of participative system design [12] the customer is included directly into the requirements and design tasks, although technical complexity is hidden inside a black-box and presented to the customer in a digestable format. The end-user is modeled as the triggering entity of example work-flows and gets the chance to edit and correct these work-flows and its associated data structures directly. Generalization is done in a stepwise fashion and makes sure that the user-model remains concise and correct.

3. Proposed Solution

On the one hand we propose a new flexible approach in the early phases of a development process. The reference model creates a new way for a flexible interplay between requirement analysis and system design. On the other hand the reference model includes a new customer-oriented modeling tool kit which explains the system considering examples of real world scenarios with animation. It should be noted that our approach, based on a new view of customer-developer interaction, does not replace existing life-cycles, notations, or tool environments but is designed as an add-on that improves end-user participation and model evaluation. Furthermore, it is compatible with most technical approaches and internally based on well known notations and tools, thus providing an interface to most life-cycles and development processes.

3.1 Reference model as process model

The new flexible approach in the early phases of the development process supports requirement analysis and system design, not stepwise successive but rather in cooperative interplay. The fundamental elements included in our reference model can be seen in Figure 3. This figure reflects the external, customer-driven view.

The reference model is composed of an outer and an inner layer. The outer layer is simply a representation layer (GUI). The inner layer is used to provide the customer with a visual description of the inner mechanisms that drive the system under development, based on examples presented as animation sequences. This will help the customer to understand why the system behaves in a certain way and moves away from a pure black-box approach that offers a GUI only. Please note that the description of the mechanism has to be a simplified and example-driven model that does not necessarily reflect the underlying technological solution and that it may preceed this concrete technological solution (that can be added-on in a later phase).

This model does not only show the inner processes of the system to the customer but also involves the customer actively in the early development of the system and can be used for further analysis steps. With the help of the *GUI* and the *Mechanism*, a horizontal prototype can be designed in collaboration with the customer. A vertical prototype can be designed to refine the mechanism further. The customer can trigger (the visualization of) an inner process with GUI elements of the outer process and observe the system's inner workings, from his perspective, by having a look at the animation of the mechanism. Both prototypes help to identify in more detail typical work-flows (*Dynamic Elements*) as well as entities (*Static Elements*) of the system.

When talking about entities and work-flows we really

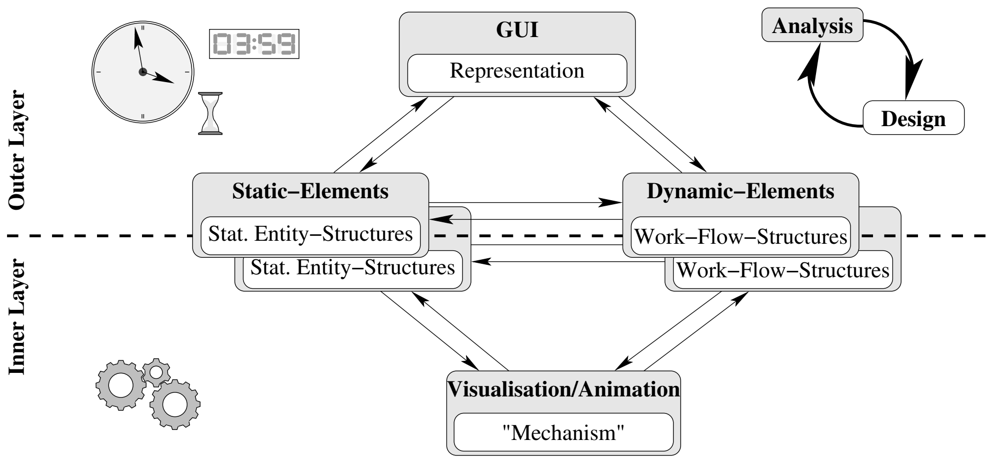

Figure 3. Customer-oriented reference model

mean entities, in the sense of instances, and not entity types (the same holds for the work-flows). This allows us to present the customer examples with traceable values, to model multiplicities, copies, recurrences, variants, and unfolded structures (e.g. trees), instead of generalized and abstracted data structures (and flows). Therefore, the dynamic generation, manipulation, and deletion of entities or subsystems, as well as the cooperation between (sub-)system instances, can be modeled explicitely and are not reduced to mere annotations in a (semi-)formal, IT specific notation that targets the type- instead of the instance-level. We strongly believe that entities (instances) are better to be understand by the customer than classes. Thus, we name our approach User Participation by EXample – *UPEX* [2].

During further iterations and refinement steps, in close cooperation with the customer, the reference model is slowly completed. With active customer participation the system analysis can promote further. This can be done by adding-on further examples and instances, e.g. to cover more unusual situations, error-handling problems, external interfaces, etc. Finally, a certain level of generalization can be achieved, we believe by the customer team itself, by classifying similar entities and work-flows and adding them also to the reference model. This will give the developer team a chance to raise the level of abstraction, starting from the basis the customer team provided, to the usual level of abstraction employed in requirements and design models, and/or to compare with already existing (internal, technology-driven) models the developer team had already developed in parallel.

3.2 Reference model as modeling kit

As mentioned before, we strive to utilize existing notations (and development processes) wherever possible to stay compatible with the main stream in software systems development techniques. However, these notations and processes must suit our requirements, as outlined above. First explorations [11], [9] have shown that most existing notations, and especially development processes, do not fit the bill right away. This is due to the fact that we think rather in instances than on the type-level and that we target executable work-flow specifications that provide for future generalization.

Currently we look at UML and its extensions, simply based on the fact that it forms a de-facto standard in the field. However, we also want to check further notations, like extended Petri Nets and work-flow systems, to form the internal engine that will drive the future *UPEX* system. We try to develop our “Static Entity-Structures” and “Work-Flow-Structures” in such a way that the structures conform to existing and widely used notations. For “Static Entity-Structures” and “Work-Flow-Structures”, we work with real world instances well known to customers. The entity types, the abstraction of the entities (classes), are hidden for customers as they are not so familiar with abstraction (see results of questionnaire). The classes are needed by the developer to find a model which is suitable to realize the software system. So our reference model is used as a modeling kit to elicit requirements and to create a (formal) model based on customers needs.

To achieve this formal model which is consistent with the customer’s needs (and not only with the documented requirements) we simulate the behavior of the system using a prototype approach on the outer level and, on the inner level, we animate (business) processes realized within the software system. Petri Nets as an abstract and formal basis for the developer are quite good for internal background of our animation. The outside representation of an animation should be an image-driven approach, e. g. business elements effected by an inner process are visualized using images, and their interplay is animated.

For a quick creation of an user interface we have looked at paper prototypes and will develop a tool which handles an easy web-based horizontal prototype without any business functions behind. With the help of this prototype the developer will be supported to find necessary interface elements and to create navigation sequences for user workflows. The prototypical graphical interface will be replaced incrementally by a final version realized by developers. So early feedback by real users is supported.

3.3 Cooperative Development

The GUI (Representation) and Visualisation/Animation ("Mechanism") in Figure 3 show the customer a view of the system. The Static- and Dynamic-Elements are further information for the customer, but not in such a clear animated and prototypical way. Static Entity-Structures and Work-Flow-Structures are notations for these elements which do not use abstract elements (classes) but instances as mentioned before. These structures (notations) should be chosen in such a way that they are conform with or transformable with common notations like UML notations. With the help of the GUI and the Visualisation/Animation the customer can detect additional instances.

Consider for example the postal delivery in an institution. Normally the secretary writes a letter for the boss. In abstraction we have two classes, secretary and boss. In our reference model we do not speak in classes but in instances. So we have two instances, Susan (secretary) and Jeff (boss). In a first conversation the customer has explained this workflow as follows: write letter $\rightarrow$ boss signature $\rightarrow$ address letter $\rightarrow$ stamp. In our animation the customer notes "print job" is missed. Hence, the work-flow element "print letter" has been added to Dynamic Elements and the animation can be adapted. Now the customer remarks the technical employee is missing who manages the printer. So Joe, technical employee, can be added to the Static Elements. In a stepwise development with the help of the animation and GUI the Static- and Dynamic Elements can be composed. With these two structures the developer can build a formal and concrete model not only on the level of instances. In further time it is considered to develop an automatism which translates the Static- and Dynamic-Elements in a model described with common notations.

4. Working Status

We have analyzed and evaluated existing notations and tools for usage in the reference model as modeling kit. Additionally we have developed a questionnaire which was handed out to a number of local companies [8]. It explores which development and process models are used in practice, especially in small and medium-sized businesses, and what is regarded to be necessary and understandable, both from a developer and user perspective.

The evaluation of the questionnaire will lead to refined requirements for a mix of notations which can be used to describe entities and work-flows customer-friendly and to drive the UPEX as its internal engine. Based on this we will have to refine the process of matching this type of reference model to customary developer tools and notations, simply to provide the capability to feed the results of the UPEX approach into any standardized development process.

Combined with an augmented development process we also propose the usage of a glossary of terms to avoid misunderstandings. At the present time we work on a web-based glossary which explains domain specific terms for domain experts and lay people.

5. Future Work

In our future works we want to design the notations "Representation", "Static Entity-Structures", "Work-Flow-Structures" and "Mechanism" in customer understandable diagrams. There we look to UML and the studies about intuitive symbols.

An evaluation of the proposed reference model should run in two cycles. After the first evaluation the results will be integrated in the reference model and after a second evaluation the reference model should be transfered in well known life cycles and notations.

References

[1] W. Emmerich. *Engineering Distributed Objects*. wiley, 2000.

[2] I. Erfurth and W. R. Rossak. UPEX - User Participation by EXample. In *Proceedings of the Joint 10th European Software Engineering Conference (ESEC) and the 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-13), Doctoral Symposium*, pages 374–376, September 2005.

[3] C. Floyd, F.-M. Reisin, and G. Schmidt. STEPS to Software Development with Users. In *ESEC '89: Proceedings of the 2nd European Software Engineering Confer ence*, pages 48–64, London, UK, 1989. Springer-Verlag.

[4] H. Giese, J. Graf, and G. Wirtz. Modeling Distributed Software Systems with Object Coordination Nets. In B. Krämer, N. Uchihira, P. Croll, and S. Russo, editors, *Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), Kyoto, Japan*, pages 107–116. IEEE Press, July 1998.

[5] E. Kindler. A Compositional Partial Order Semantics for Petri Net Components. SFB-Bericht 342/06/96 A, Technische Universität München, mar 1996.

[6] B. Krämer. SEGRAS – A Formal and Semigraphical Language combining Petri Nets and Abstract Data Types for the Specification of Distributed Systems. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 116–125, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[7] M. J. Muller. PICTIVE – An Exploration in Participatory Design. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in com puting systems*, pages 225–231, New York, NY, USA, 1991. ACM Press.

[8] Online Questionnaire "Integration of Customers in the Software Development". `http://swt.informatik.uni-jena.de/umfrage.html`, 2005.

[9] J. Petermann. Analyse existierender Modellierungsverfahren der Softwareentwick lung unter Berücksichtigung der Kundenorientierung. Technical report, Friedrich-Schiller-Universität Jena, Jena, Germany, 2003. Studienarbeit.

[10] C. Sauer and C. Cuthbertson. The State of IT Project Management in the UK. Technical report, Templeton College, Oxford, November 2003.

[11] T. Schlegel. Analyse existierender Modellierungsverfahren der Softwareentwick lung unter Berücksichtigung der Kundenorientierung. Technical report, Friedrich-Schiller-Universität Jena, Jena, Germany, 2003. Studienarbeit.

[12] D. Schuler and A. Namioka, editors. *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1993.

[13] The Standish Group. `http://www.standishgroup.com`, 2004.

[14] The Standish Group International. The CHAOS Report. Technical report, The Standish Group, 1994.

[15] The Standish Group International. Extreme Chaos. Technical report, The Standish Group, 2001.

[16] The Standish Group International. 2004 Third Quarter Research Report. Technical report, The Standish Group, 2004.

[17] L. G. Tudor, M. Muller, T. Dayton, and R. Root. A participatory design technique for high-level task analysis, critique, and redesign: the CARD method. In *Human Factors and Ergonomics Society 37th Annual Meeting*, 1993.

[18] R. Valk. Object petri nets: Using the nets-within-nets paradigm. In *Lectures on Concurrency and Petri Nets*, pages 819–848. Springer, 2003.