# Software in Which Users Find No More Errors

Olaf Musch
Projektleiter bei einem Automobil-Finanzierungs-Unternehmen in Braunschweig

Martin Wagenleiter
Geschäftsführer von Rösch Consulting Gesellschaft für innovative Softwareentwicklung mbH, Grünstraße 11, 38102 Braunschweig, mw@roesch.com

## Abstract

*Project experience indicates that zero-defect software production may be possible at very moderate cost if software is treated as behavioural knowledge and if its development is performed as a production process with industrial precision.*

*This paper describes the experience gained from a software project at the sales financing and leasing bank of a large automobile manufacturer.*

*The project team found that a very high level of software quality (process sigma > 5, 60 times better than before) can be achieved by combining the following five techniques in a flowing software development process: (1) requirements analysis (2) UML modeling and verification (3) object-oriented software architecture (4) generators (5) automated tests.*

*We feel that all of these techniques are necessary. Had we omitted any one of them, we think we would not have reached the goal.*

*Conclusion: Because it is possible to demonstrate objectively that all user requirements are met, users will not be able to find any more errors in the produced software system. And because the process is highly automated the cost of software production is lower than it is today.*

*Vision: If a project team uses these techniques in the context of a lean and tool-supported process, even the goal of "error-free software" isn't out of reach.*

## Exposé

### 1 Key Points

The key to developing software in which users find no more errors is **knowledge**, i.e. the knowledge of the users of the software. Their knowledge is the starting point of the software production process – and it is also its endpoint: At the end of the process there must be a software system which contains and automates exactly the knowledge that started the circle, not less and not more. If software developers perform their work right, the knowledge input is equal to the software output, and the users will notice no difference. This means that they will not be able to find any error.

The presentation describes the basic idea of our approach, the steps we took and the results we achieved. Here are the key points:

## 1.1 Software is Automated Knowledge

We regard software as behavioural knowledge, which tells a computer what to do.

## 1.2 Software Development is not a Service, it is a Production Process

We treat the development of software as a production process which turns knowledge into software. It is a circle:

- The circle starts with knowledge about how to handle specific situations. The knowledge comes from problem domain experts, and it gets documented in a **measurable** way.
- Software developers then convert the knowledge of the domain experts into a language the computer can understand.
- The **automated tests** perform all the measurements that were documented when the knowledge was described. This closes the circle. If all measurements show the expected results the software is shown to contain all the knowledge that was documented in the first step.

## 1.3 Challenges

While the principle of the basic idea is simple, its implementation poses a number of challenges:

- We must make sure that the knowledge in the process is **complete** (otherwise we could not rely on our automated tests as much as we do)
- We must make sure that the measurements **can be performed** at all (otherwise we would not have a chance to automate them)
- We must make sure that all measurements happen **automatically** (otherwise it would be too costly to perform them anytime we want to)

## 1.4 Tools & Techniques

We met these challenges by introducing a number of tools and techniques:

- From psychology, philosophy (constructivism) and cognitive science we borrowed techniques for finding and documenting knowledge.
- The knowledge gets documented in the form of requirements. Each requirement is accompanied by a set of examples.

- We used UML class diagrams to integrate the knowledge from different sources. These diagrams were the first intermediate products of the process. They were tested extensively and automatically, so we knew we could trust them.
- We employed an object-oriented software architecture, for several reasons: (1) It is structured 1:1 after the UML model. So it is easy to understand and to oversee. (2) It makes all components of the software easily accessible. This is a must for automatic testing.
- We used a generator to take advantage of the repetitive structures that became visible through the object-oriented software architecture. A second generator was used for flexibility, to re-generate the primary generator whenever changes in our production process were needed.
- All tests were automated and combined into one test program. This test program could be started anytime by anybody. Within seconds, it checks the whole software system, and reports differences between the knowledge and the software.

## 1.5 On the Road to Lean Production

As a side effect we noticed that our tools enabled first steps in the direction of lean production:

- The test program was used much earlier than in the testing phase: It was already used during the programming phase.
- Programming errors were caught earlier than before. Not one of them survived the programming phase (see previous point). So, as a result, the test phase was very short because not a single error was found. All errors had already been eradicated in the programming phase, by combined the “Test First” principle with full automation.
- We replaced the normally used “End of Testing Criterion” by an “End of Programming Criterion”: As soon as the test program would report “0 ERRORS: All requirements fulfilled”, programming was complete.
- It is rather obvious that the previous point practically eliminates the testing phase from software projects.
- Those of you, who think skipping tests is irresponsible behaviour, please note the following hint: If you take a factory tour (which one of the authors did at BMW and Audi) you will be surprised how few tests are performed at the end of the manufacturing process. A modern car consists of close to 100.000 parts but at the end of the manufacturing line only a few functions are tested for a couple of minutes. The rest was tested before final assembly, we assume. At least our cars have much less errors than our software systems. So we think doing tests earlier in the process will soon become the norm, also for software.

*Message: Think again about your test factories – they might become a problem (drag on action) when you want to make your move to lean production. Lean production requires early* **testing all along the process**, *closely integrated into the process, not just at the end as a separate activity.*

## 2 Novelty of the Presentation

The idea that average people – users and developers of software – should be able to produce software that is virtually error-free is new.

Today even experts are having a hard time trying to just *reduce* the amount of errors in their software systems – so why should average people have a chance to produce better results than the experts, and aim at zero-defects?

The answer is in the process:

- The process uses the strengths of the human brain:
    - Powerful imagination
    - Precision in small contexts
- The process avoids and compensates for the limitations of the human brain:
    - Loss of oversight in complex situations (7±2) [Mil 56]
    - Unability to detect causes and effects that are separated in time and space [Sen 90]
- The process is performed in small steps, many of whom are automated.
- The process has clear performance indicators. They are computed automatically.
- The process has no loose ends: it flows.

So the process fulfills the three goals one lean manufacturer has set for its own production system: [NHS 07]

Three desired outcomes:

- first, to provide the customer with the highest quality product, at the lowest possible cost, with the shortest possible lead time,
- secondly to provide members with work satisfaction, job security and fair treatment;
- thirdly, to give the company flexibility to respond to the market, achieve profit through cost reduction activities and long-term prosperity.

Goals 1 and 3 are supported by the test program. It catches any errors, long before the product reaches the customer. Costs are reduced by avoiding repairs and rework, as are disruptions from these kinds of unplannable catastrophes. This speeds up the process and allows better planning, leading to earlier delivery to the customer.

Goal 2 is supported by the clear structure of the process which makes any progress immediately visible and which helps to remove errors before they can get into anybody's way and cause any kind of stress. Software developers can be sure that the product they deliver is free from defects. This makes them feel good and gives them a relaxed feeling.

## 3 Benefits of the Presented Method

### 3.1 Reduction of Cost, Time and Errors for Lean Methods Confirmed

The process we report about has the potential to enable the introduction of lean manufacturing principles into software development.

In general, the following benefits are reported for situations where lean methods are introduced:[Wom 04]

| | |
|---|---|
| • Reduction of cost: | -50% |
| • Reduction of time: | -50% |
| • Reduction of errors: | -90% |

The project has the following characteristics: 60.000 Lines of code (LOC) and 80 tests.

Based on a productivity assumption for "normal" software development of 200 LOC per person day (PD) the expected cost would have been 300 PDs, the duration 18 months, and assuming an error rate of 1 error per 1000 LOC, 60 errors would have been normal. In the end we needed 8 months, had expended 122 PDs and 1 error made it through the defences.

| | |
|---|---|
| • Cost was reduced from 300 to 122 PD: | -59% |
| • Time was reduced from 18 months to 8 months: | -55% |
| • Errors were reduced from 60 to 1: | -98% |

### 3.2 Close to Six Sigma

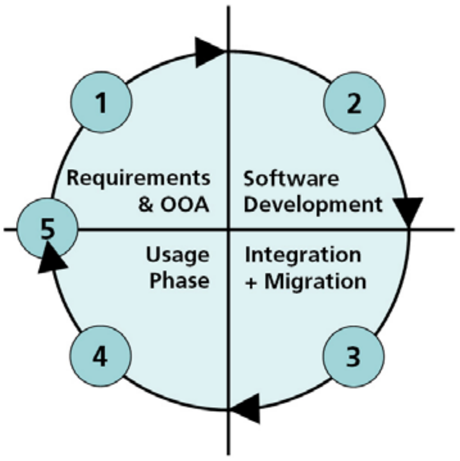
If each of the 60.000 lines of code (LOC) is counted as an opportunity for making an error, then the Six Sigma calculator of iSixSigma Magazine [Six 09] shows a Sigma value of 5.65 and a DPMO (Defects per Million Opportunities) of 17 for our project.

This level of quality is about 60 times better than the average software development, where 1 error per 1.000 LOC is still considered “normal”.

The goal for the next projects is to a Sigma value of 6, i.e. less than 3.4 DPMO.

## 3.3 A Systematic Approach to Converting Knowledge Into Software

The big picture of the process has the form of a circle. It starts and ends with the knowledge relevant to the task the software system is to automate. When the process starts, the knowledge is in the heads of domain experts. When the process has reached its end, there will be a software system containing exactly the knowledge of the domain experts. Also, there will be a verification protocol which objectively shows just this: the software contains all the relevant knowledge.

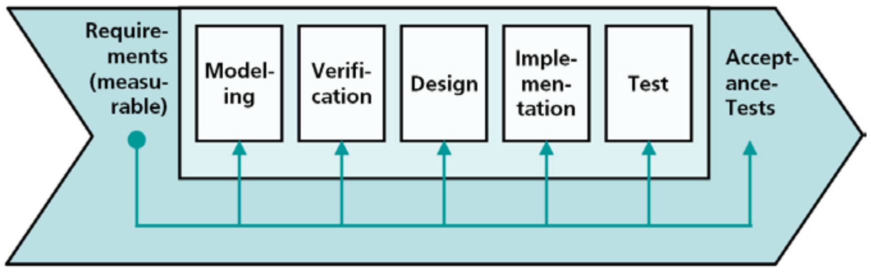


*The Development Cycle for Zero-Defect Software*

1. Requirements are collected and analyzed (OOA=Object-Oriented Analysis).
2. The software is developed.
3. The software is integrated with other systems, maybe data gets migrated. At the end of 3, the new system is ready to be used.
4. The software is used to generate value for the organization.
5. The decision for a new round of development is made. This starts a new circle.

## 3.4 Key Points For the Process

- Software is seen as **knowledge**, documented knowledge, automated knowledge. Software describes what a *computer system* must do in order to achieve a goal. If the term "software" is used in a broader sense, it may also include business processes, i.e. what an *organization* must do.
- The process is performed like a **production process**. This means that all steps are defined and preferably automated. Also, in a production process all materials must be measurable: inputs, outputs and intermediate products. In a software process these materials are knowledge; documented knowledge, in three different forms: requirements, models, and code.
- The knowledge in a software system must be described in such a way that it is **measurable** – at least qualitatively: is the desired knowledge present or absent? And even better, also quantitatively: How much knowledge is needed to run business process A? How much knowledge is embodied in software system B? The measurements must later on be turned into tests, i.e. automated tests.
- The **measurements** are performed by running the tests. All results of the measurements must be in the range of the expected results. The tests also must be fully automated.
- **Automated tests** can be started by anybody, not just by testers. So it becomes possible to use the tests long before the testing phase: even the *developers* of the software system may use them – like a second compiler. While the normal, first compiler makes sure the software is syntactically correct, the automated tests act like a second compiler: they make sure the software does what it is supposed to do, i.e. it contains the right knowledge, i.e. the software is correct.



*The Development Process for Zero-Defect Software*

*The measurements from the requirements are used at the end to perform the acceptance test, but also in the development process to test the software as it is grows.*

This process description so far has covered the main part of the process. It has shown that the knowledge is captured in the beginning and that its presence is measured at the end. This is good and necessary, but more is needed because some challenges remain:

1. We must make sure that the requirements are *technically* complete. While users are responsible for the *essence* of the requirements, we somehow must ensure that the knowledge documented is free from loose ends and unclear wording.
2. We must make make sure that all tests can be performed automatically. We must reach this goal, because we must be able to run all tests as often as we like. If tests continued to be as expensive (in terms of time and money) as they are today, we could afford only one full test per project, and we could never reach the desired level of quality (process sigma > 5).
3. Humans tend to make errors when they are working on boring and highly repetitive tasks. This is an important reason why we must reduce the amount of manual coding.

For these reasons three more techniques are needed to make the process run smoothly:

- All requirements are integrated into one coherent **UML class model**, maybe augmented by state transition models. This model is verified using the measurements that were documented along with the requirements. The **verification** not only demonstrates that the model fulfills the requirements: it also provides a basis for checking wether the model contains unspecified behaviour.
- The **software architecture** must allow easy access to every component of the software system. This means that the software architecture must be fully object-oriented. This is necessary in order to perform all the measurements automatically – which in turn allows us to run ALL tests after EVERY change, no matter how large or small the change was.
- Using **generators** allows to transfer boring tasks to automated agents. They help software developers to perform lots of boring tasks quickly and reliably.

## 4 Target Audience

This presentation targets several audiences:

- **Top Management** may learn that lean production can be applied to the production of software if certain conditions are met. Not only are the savings enormous, but the increased speed and reliability may allow whole new ways to employ software development as a strategic weapon (at least as long as competitors don't wake up to the new possibilities).

- **Middle Management** may be interested to see that several techniques, none of which is really new, can be combined to achieve results that go way beyond everything they thought was possible in the field of software quality.
- **Software Developers** may use the information in this presentation to start working in the direction described here.

## 5 Conclusion

It is possible to produce software in such a way that **users will not find any more errors** in the software produced. The key to this progress is the measurable treatment of knowledge in the software production process.

Although the software production process contains additional activities as compared to software development processes of today, it does not require additional time and money: Much to the contrary, our project has shown that **substantial savings** are possible.

While each of the five techniques used in our project has advantages of its own, it is only their **combination in a flowing process** that enabled the quantum leap in efficiency and quality we could experience in this project.

## 6 Outlook

We look forward to seeing the process being applied to the integration of multiple IT systems as well as to business processes (BPM) in conjunction with IT systems, i.e. Business-IT-Alignment.

We also feel that the techniques described here will be applicable to technical equipment like e.g. automobiles and aeroplanes.

## References

[Mil 56] George Miller: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review, 1956, vol. 63, pp. 81-97

[Sen 90] Peter M. Senge: The Fifth Discipline: The Art & Practice of the Learning Organization, ISBN-13: 978-0385260954, 1994

[NHS 07] Neil Westwood, What can the NHS learn from Toyota?, http://www.improvementandinnovation.com/features/articles/what-can-nhs-learn-toyota (29.7.2009),

[Wom 04] James P. Womack and Daniel T. Jones, Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Second Edition, ISBN-13: 978-0743249270, 2003 (1st edition 1996)

[Six 09] Process Sigma calculator: http://software.isixsigma.com/sixsigma/six_sigma_calculator.asp

## Biographies

- Dipl.-Inform. Olaf Musch has worked as project manager in the areas of banking and credit cards.
- Dipl.-Inform. Martin Wagenleiter, birthname Rösch, mw@roesch.com, Rösch Consulting Gesellschaft für innovative Softwareentwicklung mbH, phone +49 (531) 3 90 69 14.

  Martin Wagenleiter (56) studied Computer Science at the universities of Bonn and München. He graduated as Diplom-Informatiker. His professional career started at Arthur Andersen & Co as a senior consultant and continued at database company Cincom Systems before he founded his own company Rösch Consulting Gesellschaft für innovative Softwareentwicklung mbH. It specializes in innovative methods for developing software.

  Twenty years ago this meant object-orientiation and modeling. Then came software architecture, requirements and generators. Today innovation means zero-defect software production in a lean production process.

*Version: 7/30/2009 8:37 PM*