Why Software Quality Improvement Fails (and How to Succeed Nevertheless) *

Jonathan Streit
itestra GmbH
München, Germany
streit@itestra.com

Markus Pizka
itestra GmbH
München, Germany
pizka@itestra.com

ABSTRACT

Quality improvement is the key to enormous cost reduction in the IT business. However, improvement projects often fail in practice. In many cases, stakeholders fearing, e.g., a loss of power or not recognizing the benefits inhibit the improvement. Systematic change management and an economic perspective help to overcome these issues, but are little known and seldom applied.

This industrial experience report presents the main challenges in software quality improvement projects as well as practices for tackling them. The authors have performed over 50 quality analyses and quality improvement projects in mission-critical software systems of European banking, insurance and automotive companies.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software quality assurance (SQA), Programming Teams*; K.6.4 [**Management of Computing and Information systems**]: System Management—*Quality Assurance*

General Terms

Human Factors, Measurement

Keywords

Software quality improvement, Quality assurance, Change management, Industrial experience

1. INTRODUCTION

Software quality improvement plays a pivotal role for the success of IT-related business. Throughout the software lifecycle, quality deficits entail enormous costs. For instance, coding errors induce additional rework, missing documentation slows down maintenance, and inefficient algorithms increase resource consumption. They amount to billions of dollars spent each year.

Still, systematic quality management (apart from finding functional defects through testing!) does not have the self-evidence and significance in the software business that it has in other industries [20]. Although process frameworks for quality management and a variety of methods and tools for measurement exist, quality improvement often fails as

- quality-related tasks are neglected due to the needs of daily business or simple laziness.
- improvement is boycotted openly or in secrecy.
- quality management processes are applied only formally (the real problems are not reported, all indicators remain green in the management overview).
- the economic value of quality improvements is not visible and projects are stopped by management or not even approved in the first place.
- effects last only a short time and both the way of working and the resulting quality slowly drop back to their previous levels.

There are two major reasons for these failures.

Lack of Change Management.

First, the organizational and psychological aspects of a quality improvement project are underestimated or not dealt with adequately [3]. Starting such a project in an organization significantly alters the way of working and perception of the work for many employees, and thus requires change management to address their fears and needs.

However, practitioners in the field of quality measurement and improvement usually have a technical background and therefore hardly any skills and little awareness for change management issues. They tend to focus on the correctness of the metrics and tools employed and ignore other factors. This bias also dominates the scientific community: while significant research effort has been invested in the definition and validation of software metrics, the human aspects of software quality improvement remain a side issue. Change management literature, on the other hand, addresses mainly people with a management background and lacks IT-specific insights. It is thus of paramount importance that software engineers start to understand and apply change management and develop methods tailored to the software business.

*This work has partially been sponsored by the German Federal Ministry of Education and Research in the project Quamoco (01IS08023F).

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

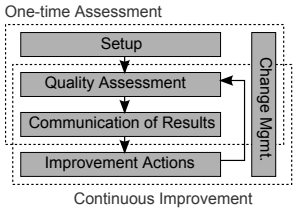

Figure 1: Phases of a quality improvement project

Economic Perspective.
A second reason for failure is ignorance of the economic reality. A quality improvement project requires investment and therefore must be able to produce and provide evidence of monetary benefits. Otherwise, it will not survive. Unfortunately, economic thinking is—similar to change management knowledge—rare in the software community.

1.1 Industrial Experience

itestra GmbH is an independent consulting company in the field of software analysis, optimization and modernization. The authors have performed over 50 quality analyses and quality improvement projects in mission-critical software systems of European companies, some of them among the Fortune 500. Presented with the analysis results, system owners have in many cases invested in quality improvement projects—in contrast to the popular belief that quality and reengineering do not receive funds. The results achieved include CPU time reduction of 30% to 80% (with similar savings in hardware and licence cost) and removal of up to 30% of unused code. See section 6 for details.

1.2 Scope

This paper presents the main challenges in software quality improvement projects as well as practices for tackling them. It applies both to one-time quality assessments and to continuous improvement in a development organization (see figure 1). The focus is put on issues specific to software quality and excludes generic change management advice. Design and introduction of processes for quality management, such as included for instance in CMMi [18], are not considered in this paper either. Extensive literature on both fields exists (see section 2) and is recommended for the implementation of a quality improvement project.

1.3 Outline

The remainder of this document is organized as follows. Related work is listed in section 2. The next sections describe the relevant stakeholder groups (3), common attitudes and fears that occur within those groups (4.1) and their reactions (4.2) as well as shortcomings within the quality improvement project itself that endanger its success (4.3). Section 5 provides approaches for dealing with those challenges. Finally, field experiences with the proposed best practices are reported in section 6.

2. RELATED WORK

Methods and standards for *software quality measurement*, such as the ISO 9126 [1] or PSM [13] focus on technical issues and the validity of the proposed metrics. Experience reports on software measurement that consider stakeholders' reaction, such as [6], are scarce.

In the field of *software process improvement (SPI)*, a variety of process frameworks and maturity models such as CMMi [18] or ISO9000 [2] and improvement paradigms such as GQM [4] and Lean [17, 21] exist. Corresponding guidebooks [5, 7, 9, 19] are available and do provide some concrete advice regarding human issues. Nevertheless, a systematic analysis of the relevant stakeholder groups with their attitudes and fears and the challenges arising out of them is usually missing.

The dynamics of change projects in general have been thoroughly analyzed in *change management literature* [10, 11]. Many of these findings and best practices also apply to software quality improvement projects, for instance the importance of developing and communicating an attractive vision for the project. However, as most recommendations are rather generic, they require additional effort and experience on the part of the individual implementing them and miss out important details specific to software organizations and software quality.

3. STAKEHOLDERS

In the following we characterize the relevant main stakeholder groups. Obviously, this classification and the attitudes attributed to each group are a strong generalization and both individuals and organizations may well deviate from it.

3.1 Senior Manager

Organizational role. The primary task of senior management is to ensure the long-term vitality of the organization (rather than short-term project and contractual concerns and pressures) [18].

General attitude towards quality improvement. Quality is an important issue for senior managers, as they are aware of the economic impact of quality deficits. They are, in many cases, the initiator and sponsor of a quality improvement project and their support is crucial for its success.

In some cases, a manager already has an opinion (e.g., one software system is to be replaced). Their goal in the quality improvement project is receiving confirmation from an independent expert or having an external party to announce unpleasant truths.

Knowledge and skills. Senior managers are accustomed to an economic, high-level perspective that summarizes data on lower levels using KPI. Contrary to popular belief, senior managers often have a good understanding of technical issues. They do see the significance and impact for the business, even if they cannot capture all details. Their decisions may however be influenced by opinions and data from the staff on which they rely.

3.2 Quality Assurance Manager

Organizational role. Quality assurance managers are responsible for defining the quality assurance processes and guidelines of an organization and monitor adherence.

Quality assurance managers often have a weak position in their organization, because they are outside the standard hierarchies. In some cases they only have an advisory function. In other cases they do control company-wide compulsory standards, but still do not have influence on budgets or the right to issue direct instructions. In many organizations, quality assurance managers have a reputation of complicating other peoples' work through additional requirements and

processes to be followed. The benefit of these processes is not acknowledged.

General attitude towards quality improvement. Quality assurance managers are usually easy to convince of a quality improvement project and its use. However, their approval is only a minor success factor.

Knowledge and skills. Quality assurance managers usually have a good understanding of the technical side of software quality, but sometimes lack the economical perspective and change management skills to achieve an enduring improvement.

3.3 Project Leader

Organizational role. Project leaders are responsible for the development, maintenance or operation of a software system and direct a development team. In most cases project leaders find themselves in a "sandwich" position: they are given instructions by their managers, and have themselves the task to sell and enforce them within their team.

General attitude towards quality improvement. Project leaders are focused on the success of their project. Time and budget pressure and (often urgent) customer requirements have a strong influence on their decisions.

Many believe that quality is in competition with those constraints (the so-called magic triangle of time, budget and quality) and give it a rather low priority, although quality problems can quickly compromise schedules and dramatically increase costs.

Other project leaders recognize the value of quality measurement and quality improvements. Even for them, however, it is sometimes difficult to incorporate the necessary tasks because of inflexible project and budget plans that force them to postpone quality-related tasks until they can be incorporated into a larger development task.

Knowledge and skills. Many project leaders started as developers and have acquired project management skills from practical experience and some additional training. Not all are familiar with management tools such as using KPI.

3.4 Development Team

Organizational role. The development team is doing the actual development, maintenance or operation work.

General attitude towards quality improvement. While some quality-related tasks, such as documentation or removal of guideline violations are considered a burden, many developers do feel pride for their work and intend to produce quality software. Their attempts to get management approval for, e.g., a larger refactoring however often fail.

An explicit quality improvement project is usually initiated by management, not by the developers. They are rather skeptical towards measurement, feel observed by it and consider it additional effort that does not provide them with new insights.

Knowledge and skills. A significant amount of developers does not have a degree or a similar higher education in computer science. A deeper understanding of software design principles or software metrics can therefore not be taken for granted.

3.5 Assessor

The assessor performs the quality assessment, communicates its results and recommends actions.

Organizational role. In most cases external consultants or quality assurance managers take the assessor's role.

General attitude towards quality improvement. It is the assessor's self-interest that the quality improvement project generates visible benefits, as his or her performance will be judged by it.

Knowledge and skills. Assessors usually have a good overview of technology issues. Knowledge of change management methods and software economics are less common, although they are crucial for the quality improvement project's success.

4. KEY CHALLENGES

4.1 Stakeholders' Attitudes and Fears

This section lists and explains attitudes and fears that frequently occur within the mentioned stakeholder groups. Fear is a strong stimulus for human behavior, and handling stakeholders' fears is one of the main topics in change management. Fears are not necessarily rational and legitimate, but might also stem, for instance, from missing knowledge.

4.1.1 Fear of Being Rated

The development team and/or the project leader are often scared that the quality improvement project will be used to rate their personal performance and that this might have negative consequences for them. Besides a general aversion to being criticized, employees fear increased pressure, the loss of a salary bonus or layoffs. Even a quality improvement project that attests them good performance might lead to increased pressure in the future, as it makes performance measurable and management expectations are likely to raise.

4.1.2 Feared Loss of Reputation

Often, there is resistance from people who believe themselves as experts in a certain field. This may or may not be true—a long history of experience does not necessarily mean somebody has fully mastered, for instance, a certain technology. A new concept (e.g. the use of Java in the eyes of a C++ expert) may invalidate their achievements, return them to the status of novice in the new field and threaten their authority towards others. People defend even ridiculously complicated practices, just because they are familiar with them or because they bear their handwriting.

A special manifestation occurs when people are asked by their manager to give an opinion on something they do not know. In this case, many prematurely produce a negative judgment ("not relevant", "don't think this will work"). Admitting they are not familiar with the issue would damage their reputation.

4.1.3 The "Fortress"

Some employees have actively "built a fortress" around a software system they are responsible for. They keep a monopoly on the knowledge and decision making, avoid to write or update documentation, claim that the system is highly critical and complex and that only they can perform maintenance tasks on it. Visible quality deficits, such as high resource consumption or manual intervention required during operation, are misinterpreted as proofs of this criticality or complexity.

The fortress is, of course, endangered by a quality improvement project, which usually involves an independent

assessment. This assessment might either show that problems are simply caused by low quality, or on the other hand show that the system is not as complicated as thought and thus that also other people could maintain it. Both is feared by the employee in his or her comfortable monopoly position.

4.1.4 Fear of Increased Workload

The development team and/or project leader often consider the quality improvement project a burden because of the additional work they expect from it. Additional tasks range from data collection, the analysis of measurement results, the identification of false positives and the removal of quality defects to the need to defend one's work against "bad" evaluations or to work with more accuracy in the future.

4.1.5 Idleness

Change in general is likely to meet skepticism from a large group of the population. From a very local point of view, change implies (in the beginning) additional effort and stress, and many people try to avoid that.

The more established a routine is, the stronger will be the reluctance to change it. Therefore, resistance is often particularly strong from people who have worked at the same company, in the same project or with the same technology for a long time.

4.1.6 Lack of Trust in Assessment

The development team and/or the project leader often do not believe that their performance is measured in an appropriate and correct way. Frequent critics include:

- *An outside person can never understand all details of the system.* Often, the context and its requirements to the system are claimed to be unique.
- *The assessors lack a thorough knowledge of the technologies used and therefore cannot understand the system.* This argument is usually put forward in the context of legacy technologies, such as COBOL or PL1 programs, but may also be used in other enclosed communities such as ABAP/SAP® development.
- *Automated measurement cannot consider all relevant aspects.* A counter-example where the metric measures wrongly is always found.
- *Code or document inspections can only cover a small part* and thus produce a rather random result.

4.1.7 Lack of Understanding

Many quality problems and/or metrics are not understood by all stakeholders due to a lack of knowledge, capacitation or thorough explanation.

For instance, [6] describes the difficulties explaining the negative aspects of global variables to COBOL programmers (who have never experienced the advantages of working with local variables and take global variables for the normal and only concept).

As a consequence such quality problems may, when indicated by an analysis tool, even be considered false positives. For instance, advanced cases of automated data flow tracing are difficult to understand and verify manually, even for experienced users.

4.1.8 Inability to Correct or Do Better

Stakeholders not knowing how to deal with a problem may deny the existence of the problem resp. the fact that a particular finding constitutes a problem.

For instance, the authors have repeatedly been told that certain code clones were legitimate because of "clones in the business domain". The individuals claiming this were obviously unable—for a lack of skill or training—to develop a parameterized design that served both use cases, and thus repudiated the criticism.

4.1.9 Focus on Unimportant Quality Aspects

When thinking about quality, developers are sometimes focused on rather unimportant quality aspects, while neglecting crucial other issues.

For instance, they proudly design a very generic, parameterizable, layered software architecture in order to increase flexibility at a point where no flexibility is needed. At the same time they neglect the fact that this architecture is larger and thus more costly to maintain.

4.2 Stakeholders' Reactions

This section describes the reactions that occur as result of the attitudes and fears explained above. These reactions endanger the success of a quality improvement project.

4.2.1 Silent Boycott

One of the most frequent reactions to a quality improvement project is to let it "starve" without openly boycotting it. Data that should be provided regularly or even proactively are delivered only upon request; forms are filled in minimalistically, such as ending all code reviews with "OK".

A quality improvement project fed with too much invalid data will not be able to produce valuable insights. This strengthens critics even more.

4.2.2 The Inerrable Expert

People who have worked with a certain system or technology for a long time often consider themselves experts in the field and are regarded as such by others. When an analysis of their code is announced, they usually respond that such a project is completely futile as everything is already in optimal condition. The assessment however reveals major improvement potential: the so-called experts have accumulated a lot of detail knowledge but lack a broader view and have not followed the newer developments.

For instance, COBOL programmers often claim that their programs are highly optimized for performance. However, this optimization is based on special coding patterns or assembler code. Efficient algorithms and data structures such as hash maps still allow significant speedup, but are unknown to most COBOL experts.

4.2.3 Putting Forward Mock Reasons

People fearing changes will often put forward reasons that seem technical issues, but that in reality are mock reasons to avoid the change.

For instance, a development team fearing an assessment claimed that their system handled highly confidential data, and that therefore the source code could not be given to an external party for assessment. However, confidentiality was only required for the data, not the code handling it.

Fixed release plans, while constituting a valid constraint in some cases, are also frequently used as a mock reason: quality improvements can supposedly not be implemented or deployed because developers or testers are occupied with functional changes, or because the correctness of a near release is claimed to be threatened by additional modifications. At the same time, quality improvements are postponed until functional changes are due in a component in order to save testing effort.

4.2.4 Adaption to a Metric

People usually adapt their way of working when they are monitored for a longer time and these measurements have consequences. This is perfectly normal and rational behavior. However, it may lead to the development team and/or the project leader concentrating on the particular aspects that are being measured, and neglecting or even sacrificing others that also are important but not captured by a metric.

For instance, a development team stopped the use of "TODO" markers when code was being searched for these markers by an automated metric. Instead, they started using other words to mark open issues. The actual goal, to measure and reduce the number of open issues, was not achieved.

This effect is particularly strong when there is a reward or punishment directly connected to the measurement.

4.2.5 Denying an Assessment's Value

After a quality analysis has been performed by external assessors, people from within the team sometimes doubt or deny that the analysis has produced insights that were not known before or could not have been achieved without external help.

Indeed, the team usually experiences many of the prevailing quality problems in their daily work. However, they seldom find a way to systematically tackle them and back down to resignation and sarcasm. An external assessment is needed to explicitly name the problems as such and to provide objective data and an action plan for improvement.

4.3 The Quality Improvement Project Itself

This section describes frequent shortcomings in the realization of the quality improvement project itself.

4.3.1 Missing Business Case

The lack of a business case for quality improvement is one of the most severe and at the same time most frequent shortcomings. Proposed actions are claimed to "improve maintainability" or "avoid defects", but an estimate of how much effort resp. money they will save is not provided (and believed to be impossible to calculate). At the same time, the cost of quality improvement actions is easily visible. Both project leaders and senior managers therefore often decide against suggested actions.

Some quality improvement projects not only fail to point out their economic value, but simply do not provide one. For instance, they produce documentation that is not used and maintained by the developers.

4.3.2 Too Many False Positives

Automated tools that search code for quality deficits often report a large number of false positives, at least if not configured appropriately. If results are shown to a team at this stage, this will dramatically reduce their trust in the analysis as a whole (see 4.1.6). Furthermore, false positives increase the cost of quality defect removal, as all findings have to be analyzed first.

[6] suggests a false positive rate of less than 20% in order for a tool to be usable in practice—a threshold that requires quite some adaption and configuration with many tools.

4.3.3 Too Much Data

Often, too many findings or measurement results are presented at the same time and without a clear purpose. As a consequence people will have difficulty identifying the important results and drawing conclusions. They quickly become annoyed by what they perceive a uniform mass of data. The same may happen in a continuous measurement program when data is reported too often [5].

4.3.4 Self-Confidence of Assessors

The individuals who assess the quality of a software system they do not know need strong self-confidence. In a rather short time they have to get both an overview of the system and insights into important details. They have to generalize and extrapolate, to decide where to dig into and what to neglect, and to draw conclusions from insecure or partial information. At the same time, the development team will often doubt their ability to produce valid results (see 4.1.6). All this may give the assessor, in particular novices, the feeling that they cannot perform their task adequately. They may be tempted to adopt developers' views (who claim to be experts for their system) without sufficiently challenging them.

5. BEST PRACTICES

This section presents well-proven best practices that help to overcome the above challenges and thus to ensure the success of quality improvement. Best practices are categorized by the project phase in which to apply them (see figure 1).

5.1 Setup

The following best practices should be considered when planning a quality improvement project.

5.1.1 Provide a Business Case

A quality improvement project must be able to explain and quantify its cost and benefits as well as the duration after which a positive return on investment (ROI) is expected. Estimates should be used where no exact figures are available.

5.1.2 Obtain Management Support

Support from senior management is crucial for the success of any change project. This includes not only the budget and a general commitment that the project is wanted, but also the readiness to draw consequences from the results and take action—potentially against resistances in the organization.

The necessary investment and support, duration and implications of a quality improvement project should be explained to the sponsor very clearly from the beginning on. If the sponsor is not willing to bear this full effort, the project cannot be successful anyway.

5.1.3 Set Common Goals

Defining the goals of a quality improvement project together with all stakeholders and assigning everybody a task

makes people feel involved. As a consequence, they will show less resistance and more dedication than for "somebody else's" project. It helps when management and project leader introduce and explain the quality improvement project in front of their team (in contrast to an external assessor doing so). Thereby they show their commitment in public and will back down less easily during the project.

5.1.4 Respect the Context

The context of a quality improvement project may differ from one company or situation to another and should be respected when planning the project. For instance, team cultures may vary from a tight control of developers to a very trustful atmosphere. In the first case, developers are more likely to adapt their behavior to a metric and more attention has to be paid to this issue, while in the second case developers might take too much monitoring as an insult.

5.1.5 Respect Employees' Privacy

Data that can be used to rate individual performance should be collected very carefully, if at all. It should be investigated beforehand whether an approval from employees' representatives is required.

[19] suggests, as a reaction to the need for privacy, that ownership of the measurement data should be attributed to the development team. All reporting to management is to be approved by them.

5.2 Contact with Stakeholders

The following best practices apply every time communication with stakeholders takes place.

5.2.1 A Benefit for Everyone

Explain to all stakeholders how they can benefit from the project. If possible, this benefit should be a personal one (such as less time spent on debugging) rather than a general benefit for the company.

Note that it is necessary to analyze the motivations of each stakeholder group thoroughly, as they may run deeper than the obvious and rational. For instance, a manager responsible for a software system may consider it beneficial that the system uses a lot of CPU resources, because this implies a large budget and thus prestige for him.

5.2.2 Be the Developer's Friend

The people who will be responsible for improving software quality are the developers. It is crucial that they understand and accept the analysis results and the conclusions drawn. Therefore, the assessors should try to gain their trust. The following approaches have proven to be useful:

- Helping the developers to solve problems they suffer from. For instance, the assessors may convince management to allocate time to refactorings the developers consider necessary or make a case for better equipment.
- Tackling the issues raised by the quality analysis actively and immediately, so that they are not perceived as another burden put on the developers' backs. This can be achieved, for instance, through an early improvement pilot (see 5.4.2) during which the assessors demonstrate how to deal with a problem.
- Criticizing not only problems in the code but also problems that stem from management, such as inadequate processes.
- Listening to developers' opinions and feedback and respecting them.

5.2.3 Look Forward, Not Back

Many people will take negative assessment results as an offense to their work and person. It is therefore important to explain clearly and before the presentation of actual results that:

- *It is to be considered normal (although not desirable) that software systems contain quality flaws.* Most software development projects suffer from time pressure and changing requirements, which usually compromises quality. Also, the quality of a software system is known to degrade over time [14] if no explicit counteractions are taken.
- *The goal is to improve in the future, not to blame anybody for the past.* It helps to accept rationale for past decisions ("Copying code to create a variant was recommended by COBOL guidelines."), as long as the team accepts that there is a problem with it now and is willing to do differently in the future. In some cases, the developers responsible for quality flaws have left the organization anyway. The rationale provided may also point to improvement potential in the development process ("I have to copy the code when I need a variant because I do not have the access rights to modify the original in the common library.").

5.3 Communicating Results

The following best practices should be considered for the communication of assessment results.

5.3.1 Present Intermediate Results

Separate presentations are needed for different audiences at different times.

An intermediate presentation and discussion with the development team should take place when a first set of results is available. This fulfills several purposes:

- *Getting feedback on the validity of the findings.* Developers may confirm the findings and provide additional information, such as further examples, the history of a problem, rationale for a certain solution or hints on deficits in the development process that lead to quality flaws. In other cases developers will disagree with the findings or the conclusions drawn. This gives the assessors the opportunity to improve or correct the analysis before presenting a final version (if the finding was wrong), or to provide a better explanation (if the developers misunderstood the problem).
- *Reducing resistance.* A foreign judgment seems threatening to most people and results in refusal. If people feel that their opinions have been heard and accounted for during an intermediate presentation and thus that they have contributed to the assessment, resistance is lower.
- *Bringing forward criticism and resistance.* Adaption to change (such as developers having to change their

way of working) usually follows a process that can be described by the five stage Kübler-Ross-cycle [12]: Denial, Anger, Bargaining, Depression and Acceptance. If developers are first confronted with results at the end of an analysis, they will be in Denial or Anger stage and thus show strong opposition at a time when management presentations and decisions about follow-up actions are due. This can jeopardize the success of the project. It may therefore be wise to trigger the cycle early and in a controlled way (through an intermediate presentation with the developers), so that they have overcome the Anger stage when final results are there.

A presentation and discussion with the sponsor of the project, usually a senior manager, should take place before results are spread to a broader audience. This allows to discuss implications early and prepare for company-internal struggles.

5.3.2 Adapt the Argumentation to the Audience

Different stakeholders need different ways of argumentation. For management audiences, the impact of quality deficits on costs and risks is most important. Developers will be more open to an argumentation based on technical problems that arise from quality deficits as well as the impacts on their daily work. This can be achieved by using different views of a quality model for different audiences, such as a hierarchy of cost items for management and a hierarchy of technical issues (see [16]) for the development team.

"Buzzwords" and management vocabulary, such as "ROI", should be avoided for developer audiences.

5.3.3 Provide Different Levels of Detail

Analysis results should be available on different levels of aggregation, with the topmost summary fitting on one page or slide.

A drilldown should be available, i.e., it should be possible to trace aggregated results back to the concrete findings that lead to the result. Example:

Level 1: "low" rating for maintainability

Level 2: Average module length 1021 LoC

Level 3: List of modules exceeding threshold of 500 LoC

While a drilldown is crucial for gaining acceptance, experience shows that it is not used so often in later stages, when stakeholders have gained trust in the aggregated results.

5.3.4 Use Both Metrics and Code Samples

Code examples significantly improve credibility, even for management audiences. They also show the team that the assessors master the programming language and technology used. Note that code samples from the actual system under scrutiny have a much bigger effect than generic examples. Metrics results, on the other hand, prove that the examples shown are not only isolated cases.

5.3.5 Provide Benchmarks

The comparison to a benchmark is important because people want to know whether their results are "good" or "bad", and often cannot tell from the bare numbers.

Both the comparison with a very bad result ("You surely can do better than project X") and the comparison with the benchmark ("This is what the best achieve") can help to motivate a team.

5.3.6 Prepare, Filter and Highlight Data

Data should be presented in a way that allows the audience to easily understand it and to draw conclusions. Relevant findings and critical modules should be highlighted, irrelevant ones hidden. A traffic lights color scheme allows to identify problems fast.

A good approach for finding an appropriate way of presentation is to imagine or watch different stakeholders when shown the data. If, for instance, the first thing they do is to scan the list of modules for results above a critical threshold, it might be a good idea to directly present that list in a sorted or filtered way.

5.3.7 Explain the Effects

For management presentations, it is crucial to show the economic effects of quality deficits and improvements as well as the connection to the business goals. Ideally, a case study from a similar situation and similar domain is available.

But also for the development team, the effects of quality deficits should be explained. It should be made clear that the criticized aspects are *not* a matter of taste or beauty, but have significant impact on operation, maintenance etc. An activity-based quality model [8] provides the necessary links. The argumentation should be supported by measurements and examples, such as the defect history for a component that is found to be difficult to understand.

5.3.8 Do Not Be Smart

As explained in 4.1.2, people do not like to be taught in their own field. Unfortunately, presenting and explaining quality deficits is exactly this. Several strategies serve to reduce resistance:

- *Let people develop the interpretation by themselves.* The result presentation must provide them with all necessary information and hints, but not tell them the interpretation. When they figure out the last step by themselves they will both accept and remember it better. For instance, a diagram showing a low productivity for the system analyzed and higher productivity for other systems will probably be enough for the audience to conclude that they should do better.

- *Let the peers do the arguing.* This is useful for dealing with people who are either very convinced of their opinion or put forward mock reasons. In both cases rational arguments brought forward by an external assessor will be of no help. Other team members, however, may manage to convince their peers. An opportunity for this is to mention the issue in a large meeting with all the team present [6].

- *Let people save face.* Do not prove anybody wrong in public but give them an opportunity to change their opinion and declare the correct fact or interpretation themselves.

5.3.9 Provide Action Hooks

The aim of quality measurement is quality improvement, and the findings presented should help to fix problems. In order to achieve this, a drilldown is needed that allows to trace quality ratings back to the locations of concrete deficits in the code. Furthermore, the assessment must clearly explain which actions are required to correct the flaw ("extract

blocks from oversized method", "break circular dependency by introducing an interface" etc.).

Such concrete advice improves the acceptance of automated measurement in the development team, providing the analysis directs them to places they consider critical themselves and not only entails annoying code cleaning tasks.

5.3.10 Prepare Answers for Frequent Questions

The same questions and arguments are asked resp. put forward throughout different companies and contexts. An assessor who has answers, materials and counter-examples ready can immediately invalidate the critics.

For instance, the frequent doubt "Do the assessors have enough knowledge of the system and its technology" (see 4.1.6) can be countered as follows:

- Many important quality aspects, such as documentation or identifier naming, can be easily assessed without thorough knowledge of the programming language.
- The assessment will involve the developers in order to ensure that all necessary details are respected.
- It is even helpful that the assessors do not have the same background as the development team! How can you expect someone to solve a problem if they think in the same ways as the people who created it?

5.4 Planning Improvement Actions

The following best practices apply when planning the implementation of improvement actions.

5.4.1 Schedule Action Early

Implementation of suggested actions should begin immediately after the analysis. In order to ensure this, analysis and improvement actions should not be treated as independent issues, but as two parts of the same quality improvement project. This should be reflected in all communication and planning from the beginning on.

Starting improvement moves focus from analysis (which is usually perceived as criticizing and thus negative) to constructive action. The benefit of the implemented improvement can be skimmed off and invested in further analyses and actions. Finally, a fast implementation offers the opportunity to concretize estimates for effort and benefits.

First implementation steps can be made in the form of a pilot action, such as removing unused code in one component of a system. An iterative approach with alternating analysis and implementation increments should be used for larger projects.

5.4.2 Demonstrate Improvement

It is often a good approach to have the assessment team implement the first improvement actions themselves. This moves focus from analysis to constructive action. It proves to the stakeholders that there is actually a way to overcome the problems and that the proposed actions are feasible. In the case of resistances or a simple inability to correct the problems (see 4.1.8), it ensures that the improvements are actually implemented instead of being postponed and "starved", and thus that the quality improvement project provides an immediate value. Note however that management often (wrongly) believes their staff could correct the problems themselves and thus considers involvement of an external party unnecessary.

5.5 Continuous Improvement

The following best practices should be considered when introducing a continuous quality measurement and improvement program.

5.5.1 Start Small

A continuous measurement program should start with few metrics and—if possible—by reusing data that is already being collected.

5.5.2 Provide Enough Budget

Enough effort should be allocated to data collection, analysis and derived action. Manual quality assurance and correction of collected data is usually needed and should be allowed for, as well as investments in easy-to-use tools and processes and their maintenance over time.

[5] recommends that the effort invested into analysis should be three times as much as the effort invested into pure data collection. Otherwise, it will not be possible to draw the right conclusions and perform the necessary actions, and thus the fruits of the measurement will not be collected. In reverse, this means that for a limited budget, the number of metrics must be adapted accordingly.

5.5.3 Provide Independent QA and Support

An independent body should control that the measurement process is performed in the right way, even if at some point a development team understands the benefits and takes over responsibility.

Furthermore, [5] recommends to relieve the developers from anything except the delivery of raw data, and have a separate packaging team perform data quality assurance, correction and processing.

5.5.4 Provide Fast Feedback

A short feedback loop is essential. The easiest way to correct quality problems in a module is when the file is still open in the editor. After the file has been closed or even checked in, a modification requires more effort and is therefore less likely to be performed. After days or even weeks, a developer will have to invest additional effort in re-understanding the code. For instance, hardly any team corrects the thousands of findings indicated by the FindBugs or CheckStyle tool when run on an existing project instead of being integrated into the development environment from the beginning.

Tools that enable the developers to check some quality aspects themselves immediately, such as an integrated style checker, also allow them to correct the weaknesses before somebody else notices and thus to avoid embarrassment.

Furthermore, fast feedback shows the team that their behavior indeed makes a difference (in the good sense when results show an improvement, or in the bad sense when results show that quality continues to degrade).

5.6 Relation to Challenges

The following figure 2 provides an overview of which best practice is meant to tackle which challenges.

6. INDUSTRIAL APPLICATION

Since 2006, the authors have applied the presented practices in quality improvement projects. As a first step of such a project, we conduct a quality analysis called Software

Challenge / Best Practice	Provide a Business Case	Obtain Management Support	Set Common Goals	Respect the Context	Respect Employees' Privacy	A Benefit for Everyone	Be the Developer's Friend	Look Forward, Not Back	Present Intermediate Results	Adapt the Argumentation to the Audience	Provide Different Levels of Detail	Use Both Metrics and Code Samples	Provide Benchmarks	Prepare, Filter and Highlight Data	Explain the Effects	Do Not Be Smart	Provide Action Hooks	Prepare Answers for Frequent Questions	Schedule Action Early	Demonstrate Improvement	Start Small	Provide Enough Budget	Provide Independent QA and Support	Provide Fast Feedback
Fear of Being Rated			x	x	x	x	x	x	x	x				x				x						
Feared Loss of Reputation				x		x		x								x		x						
The "Fortress"				x		x		x										x						
Fear of Increased Workload	x	x	x	x		x	x			x					x		x	x		x	x	x	x		
Idleness				x		x	x					x	x		x		x			x	x		x	x	
Lack of Trust in Assessment			x	x					x	x	x	x						x		x	x			x	
Lack of Understanding				x					x	x	x	x		x	x		x	x		x				x	
Inability to Correct or Do Better				x			x		x	x		x					x	x		x					
Focus on Unimportant Quality Aspects			x	x					x	x			x	x	x			x		x					
Silent Boycott[1]		x		x					x										x	x		x	x	
The Inerrable Expert	x	x		x					x	x		x	x		x			x	x	x				x	
Putting Forward Mock Reasons	x	x	x	x					x	x		x	x		x	x		x	x	x				x	
Adaption to a Metric				x													x						x		
Denying an Assessment's Value	x	x	x	x								x	x		x			x							x	
Missing Business Case	x														x				x		x	x			x	
Too Many False Positives														x							x	x	x		
Too Much Data											x			x							x				
Self-Confidence of Assessors		x		x														x	x	x					

[1] Problematic reactions can, of course, be dealt with by eliminating their root cause, i.e., the attitude or fear that lead to this reaction. The entries in this part of the table, however, are limited to the best practices that directly impede or neutralize the reaction.

Figure 2: Relation between Challenges and Best Practices

HealthCheck [15], leading to a set of recommendations for improving the economic efficiency of the analyzed system. Measurement results, interpretation and recommendations are presented and discussed with both managers and developers. The owner of the system then decides on further steps.

The authors have conducted over 50 quality analyses of business information systems, comprising more than 60MLoC. About 60% of the systems were implemented using COBOL and PL/I, 20% in C/C++ and 20% in Java/C#. The domains of the systems include automotive/industry (24 systems), insurance (13) and banking (4).

For about half of the analyses we have performed, system owners have initiated an implementation of some or all of the recommendations. Among 20 customers for which we have run analyses, 12 started improvement projects and another 5 are considering to do so. 6 of the customers aim to improve system performance, 4 aim to reduce maintenance cost and increase productivity, 2 are involved in both. The results achieved include a reduction of CPU time for the programs covered of 30% to 80% (with similar savings in hardware and licence cost), redocumentation of modules considered "unmaintainable" and removal of up to 30% of unused code. In many cases, the original developers have performed the implementation or contributed to it, and recognized it as an improvement. The overall project budget exceeds 3.500 person-days of external consulting and implementation.

The fact that system owners decide to invest in quality improvement—often after years of degrading quality and in contrast to the popular belief that quality and reengineering do not receive funds—and the successful collaboration with the developers shows that the findings, their importance and consequences have been effectively communicated.

We believe that the presented best practices can promote the success of quality improvement and serve as a base for further exploration within the scientific community. In order to gain influence in the industry and to achieve appropriate quality of their products, software engineers imperatively have to familiarize themselves with change management and economics and to adapt them to their field.

7. ACKNOWLEDGEMENTS

In addition to the authors' experiences, practitioners and researchers from six organizations within the Quamoco project consortium[2], in particular Capgemini and itestra GmbH, have contributed their expertise to this report. Work has partially been sponsored by the German Federal Ministry of Education and Research in the project Quamoco (01IS08023F).

8. REFERENCES

[1] ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model, 2001.

[2] ISO 9001:2008 Quality management systems – Requirements, 2008.

[3] Carolyn Aiken and Scott Keller. The irrational side of change management. *McKinsey Quarterly*, April 2009.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Encyclopedia of Software Engineering*, chapter The goal question metric approach, pages 528–532. John Wiley & Sons, Inc., 1994.

[5] Mitchell J. Bassman, Frank McGarry, and Rose Pajerski. *Software Measurement Guidebook*. Number 94–102 in Software Engineering Laboratory Series. NASA Goddard Space Flight Center, 1995.

[6] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Communications of the ACM*, 53:66–75, 2010.

[7] Carnegie Mellon University. *IDEAL: A User's Guide for Software Process Improvement*, 1996.

[8] Florian Deißenböck, Stefan Wagner, Markus Pizka, Stefan Teuchert, and Jean-Francois Girard. An activity-based quality model for maintainability. In *Proceedings of the 23rd ICSM*. IEEE CS Press, 2007.

[9] Christof Ebert, Reiner Dumke, and Manfred Bundschuh. *Best Practices in Software Measurement. How to use metrics to improve project and process performance*. Springer, Berlin, 2004.

[10] Wendell L. French and Cecil H. Bell. *Organization Development: Behavioral Science Interventions for Organization Improvement*. Prentice Hall, 1999.

[11] John P. Kotter. *Leading Change*. Harvard Business Press, 1996.

[12] Elisabeth Kübler-Ross. *On Death and Dying*. Macmillan, 1969.

[13] John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, and Fred Hall. *Practical Software Measurement: Objective Information for Decision Makers*. Addison-Wesley, 2002.

[14] David Lorge Parnas. Software aging. In *Proceedings of the 16th ICSE*, pages 279–287, Los Alamitos, 1994. IEEE Computer Society Press.

[15] Markus Pizka and Thomas Panas. Establishing economic effectiveness through software health-management. In *1st International Workshop on Software Health Management*, Pasadena, July 2009.

[16] Reinhold Plösch, Harald Gruber, Gustav Pomberger, Stefan Schiffer, and Christian Körner. A proposal for a quality model based on a technical topic classification. In *2. Workshop SQMB*, 2009.

[17] Mary Poppendieck and Tom Poppendieck. *Lean Development – An Agile Toolkit*. Addison-Wesley, 2003.

[18] CMMI product team. CMMI ® for development, version 1.2. Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon University, 2006.

[19] Rini van Solingen and Egon Berghout. *The Goal/Question/Metric method: A practical guide for quality improvement of software development*. McGraw-Hill, 1999.

[20] Stefan Wagner, Manfred Broy, Florian Deißenböck, Michael Kläs, Peter Liggesmeyer, Jürgen Münch, and Jonathan Streit. Softwarequalitätsmodelle – Praxisempfehlungen und Forschungsagenda. *Informatik Spektrum*, 33(1):37ff, 2010.

[21] James P. Womack, Daniel T. Jones, and Daniel Roos. *The machine that changed the world: The story of Lean Production*. Free Press, 1990.

[2] `http://www.quamoco.de`