A Proactive Attitude Toward Quality: The Project Defect Model

Ben Linders, Ericsson Telecommunicatie B.V. The Netherlands, and Hans Sassenburg, SE-CURE AG, Switzerland

Published in: Software Quality Professional, Volume 7, Issue 1, december 2004.

Over the last decade, the software industry has been investing heavily in software process improvement, such as CMM(I) initiatives. But achieving higher CMMI levels does not guarantee business success. Being able to define strategic business objectives and implementing a derived product development strategy with a proactive attitude toward quality determines whether one is successful in the long run. Although the importance of quality is generally not questioned, during project execution the focus of project managers tends to shift toward meeting functional requirements, schedule, and budget. It is only during the late testing phase that quality becomes important again.

This article focuses on the importance of quality, where the existing approach toward dealing with quality in the late testing phase is criticized. The author discusses how quality can be managed proactively, using an example from industry. Ericsson R&D Netherlands defined a project defect model and ran a pilot project to manage quality throughout product development. This article describes the model, its implementation, and the results. Although it is not easy to measure, track, and steer quality during a project, this article shows that it is possible. Combining the quality approach with existing management methods enables an organization to manage all project core dimensions.

Key words: business benefit, defect management, goal question metric, measurement, MTB, product development strategies, quality predictions

INTRODUCTION

The software industry has been investing heavily over the last decade in software process improvement. Broadly accepted models such as the Software Engineering Institute's Capability Maturity Model (CMM®) and its successor, the CMMI®, are used to define and

institutionalize software development processes. Progress is reported on an ongoing basis, and an increasing number of software manufacturers have reached higher capability maturity levels. These process improvement initiatives have paved the way for taking the next step in making software engineering the discipline its founders envisioned. What is the next step? Isn't CMM® level 2, level 3, or even level 4 enough? Some say that it is not. Achieving a certain maturity level is not a guarantee for business success. Being able to define strategic business objectives and implementing a derived product development strategy with a proactive attitude toward quality determines whether one is successful in the long run.

Many software manufacturers struggle with quality; it is often not more than paying lip service. Although its importance is generally not questioned, during project execution the focus of project managers tends to shift toward meeting functional requirements, schedule, and budget. It is only during the late testing phase that quality becomes important again.

This article emphasizes the importance of quality independent of which product development strategy is chosen. Further, the existing approach toward dealing with quality in the late testing phase is criticized. The authors discuss how quality can be managed proactively, using an example from industry. Ericsson R&D Netherlands defined a project defect model and ran a pilot to manage quality throughout product development. Based on successes, many projects now use the model. This article describes the model, its implementation, and the results. Although it is not easy to measure, track, and steer quality during a project, this article shows that it is possible. Combining the quality approach with existing management methods enables an organization to manage all project core dimensions.

PRODUCT DEVELOPMENT STRATEGIES

It is commonly accepted that the four core dimensions of project management are: functionality (what), quality (how good), cost (how much), and time (when). Project management deals with balancing these four dimensions in such a way that business requirements are met. These business requirements are documented in a so-called business case, which has twofold objectives. The first objective is expected revenues and costs related to the product innovation, and the second objective is the release criteria together with their relative priorities. The business case will definitely raise the question: "What has more value: time-to-market, unique product features, high quality, low development, or operational cost?" A software manufacturer must recognize that it cannot be all things to all people and must

focus on distinguishing itself in the marketplace. Possible product development strategic orientations are:

- **First mover**. This orientation focuses on getting a product to market very fast. This is typical for software manufacturers involved in rapidly changing technology or products with rapidly changing fashion. Pursuing this strategy typically leads to tradeoffs in optimizing functional requirements, development cost, and quality.
- **Lowest development cost**. This orientation focuses on minimizing development cost or developing products within a constrained budget. This orientation occurs when software manufacturers are developing under contract for other parties, where a company has severely constrained financial resources. It involves tradeoffs with functional requirements, time-to-market, and quality.
- **Unique product features**. This orientation focuses on having the highest level of functional requirements or product features, including aspects such as the latest technology and/or product innovation. It involves a trade-off of time-to-market, development cost, and quality.
- **Highest quality**. This orientation focuses on assuring high levels of product quality (reliability, safety, and so on). It is typical of industries requiring high quality because of the significant costs after product release to correct a problem (for instance, recalls in a mass market), the need for high levels of reliability (for instance, the aerospace industry), or the significant safety issues (for instance, medical devices). It corresponds to the orientation of minimizing operational cost. It involves a trade-off of functional requirements, time-to-market, and development cost.

Which strategy should be chosen? Card suggests that the number of potential buyers and the competition level together determine the kind of strategy that is the most profitable in the long run (Card 1995) (see Figure 1).

		Competition level	
		Low	High
Buyers	Few	Unique product features	Lowest development cost
	Many	First mover	Highest quality (Lowest operational cost)

Figure 1 Model of software markets (Card 1995)

The product development strategy might change as a product matures. Moore investigated why many new technology companies started with new inventions and rapid market growth, but collapsed within the next three years (Moore 1995; see also Denning 2001 in which results of Moore's study are summarized). He explained the phenomenon by recalling an earlier model of mindsets toward the adoption of technologies. Initial success is gained by selling products to technology enthusiasts and visionaries, who are quick to grasp the implications and care less about issues such as reliability. When the market of visionaries becomes saturated, however, the attempt to sell the technology to pragmatists might fail, as they care more about stability or reliability. Moore used the metaphor of a chasm: the company leadership discovers too late that it does not communicate with the pragmatists (see Figure 2).

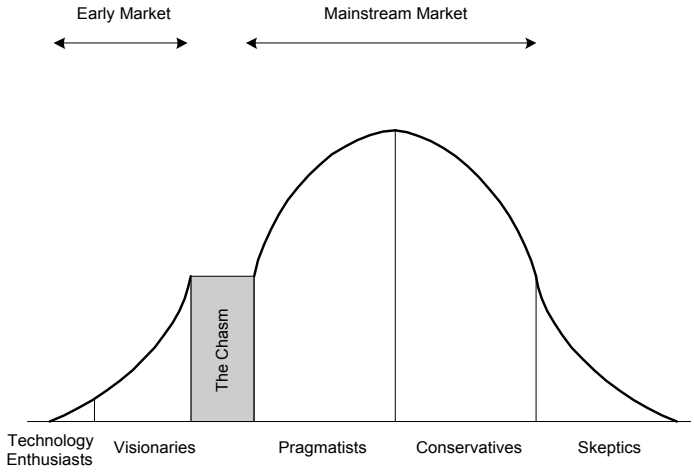

Figure 2 Chasm between the early market and the mainstream market (Moore 1995)

Moore also presented a model that shows how the project priorities or criteria shift during the evolvement of a product seen from the customer's perspective. This is illustrated in Figure 3 (Moore 1995).

Market description	Introduction	Early Adopters	Mainstream	Late Majority	End of Life
Buyer profile	Technology enthusiasts	Visionaries	Pragmatists	Conservatives	Skeptics
Importance	1. Time to market 2. Product Needs 3. Quality	1. Time to market 2. Product Needs 3. Quality	1. Quality 2. Time to market 3. Product Needs	1. Quality 2. Product Needs 3. Time to market	1. Quality 2. Product Needs 3. Time to market

Figure 3 Project priorities as a function of a product's life cycle (Moore 1995)

One can conclude that the best strategy depends on the company's capabilities (strengths, weaknesses, and core competences), market needs, and opportunities, goals, and financial resources. There is no "right" strategy for a software manufacturer, but it is important that a strategy is chosen. Independent of the chosen product development strategy, it is always necessary to manage the quality dimension. Although the customer might be less interested in quality during the early phases of a product, the software manufacturer will benefit from a quality focus. Being able to limit the injection of defects or increase defect removal in early development stages will normally lead to increased productivity and a less faulty product. In general, this effect will allow a software manufacturer to deliver a better product earlier to the market. (For more information, see also Sassenburg 2003.)

FOCUS ON QUALITY DELAYED UNTIL TESTING

Quality is normally defined as a nonfunctional requirement during the project definition phase. However, during product development the focus tends to shift toward meeting product needs, time-to-market, and budget (cost). Although nobody denies the usefulness of reviews or inspections, when time pressure increases these are normally the first activities that are skipped. It is only during testing that quality becomes important again with a strong focus on reliability only. As a result, serious attention to quality is normally seen when the project is nearing the release date. It is when integration and system tests are executed. Many software defect prediction models have been formulated to find the optimal release time for software products. These models support the trade-off between the three dimensions of cost, time, and quality during the test phase (see Figure 4).

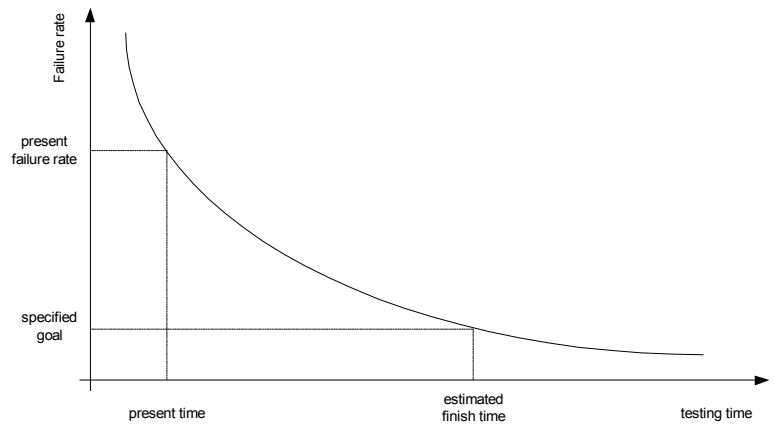

Figure 4 Basic idea of software reliability estimation modelling

Fenton and Neil (1999) and others (Gokhale 1996; Wallace and Coleman 2001) conclude, however, that these prediction models provide little support for determining the reliability of a software product. Their underlying assumptions do not reflect reality.

Research revealed that software manufacturers struggle with determining the right moment to release a software product (Sassenburg 2003). The analytical defect prediction models are either unknown or not used. Instead, in practice often a combination of the following nonanalytical methods is used to decide when a software product is "good enough" to release (RTI 2002):

- A "sufficient" percentage of test cases run successfully.
- Statistics are gathered about what code has been exercised during the execution of a test suite.
- Defects are classified, and numbers and trends are analyzed.
- Real users conduct beta testing and report failures that are analyzed.
- Developers analyze the number of defects found in a certain time period. When the number stabilizes or remains below a certain threshold, the software is considered "good enough."

Not knowing the quality level when releasing a software product leaves the manufacturer exposed to the following risks:

- **Unpredictable product behavior**. It is difficult to guarantee to the user(s) what the exact functionality of the product will be. This may lead to user dissatisfaction and to unforeseen, even potentially dangerous, situations.
- **Unknown operational cost**. The post-release or operational cost of the software products may be unexpectedly high. For example, the exact status of the software product and its documentation may be unknown leading to high corrective maintenance costs. In addition, adaptive and perfective maintenance activities may be severely hampered.

It is concluded here that focusing on quality only during the late testing phases should be avoided. It is not only expensive, but also very risky. A more proactive attitude is needed during the earlier development phases. In the doctoral research for release decisions, Ericsson Telecommunicatie B.V. in the Netherlands was visited by Hans Sassenburg. This company has developed and implemented a defect estimation model, used to measure and control

quality throughout the software life cycle next to budget, time, and functionality. The model is described next.

MANAGING QUALITY: A CASE STUDY

Ericsson Telecommunicatie B.V. in the Netherlands has several units. Among them are a market unit, which sells products and services to the local markets, and a research and development (R&D) unit, which is the software design center for worldwide intelligent networks product development. Software plays a very important role in Ericsson's products. This importance has led to large investments into software process improvement over the last decade. In 1995, the Ericsson R&D unit in the Netherlands was the first Dutch organization to reach CMM level 3. Since then, further investments have been made to increase its software development capability. The focus of an ongoing improvement program is to align software metrics with business objectives with a strong focus on managing quality.

Metrics

Can quality be proactively managed? In the first place, the requirements for measuring quality are examined using the goal-question-metric (GQM) approach (Solingen and Berghout 1999):

Goals:

1. Control verification activities (optimize defect detection)
2. Control development activities (minimize defect injection)
3. Predict release quality of the product
4. Improve the quality of the development and test processes

There is a need for measurements in order to steer quality -- measurements usable to plan quality at the start of the project, and to track it during project phases, enabling corrective and preventive actions and reducing quality risks in a project. An additional project need is to estimate the number of latent defects in a product at release. The purpose is twofold. First, it is usable to decide if the product can be delivered to customers, or released, knowing the quality. Second, it helps to plan the support and maintenance capacity needed to resolve the defects that are expected to be reported by customers. Finally, it should be possible to have quality data that can be analyzed together with the applied processes, and the way a project is organized. This analysis provides insight into process and organizational bottlenecks, and, therefore, enables cost-efficient improvements.

Questions:

1. What will be the quality of the released product?
    a. Per requirement?
    b. As perceived by customers?
2. How good are inspections?
    a. How effective is the preparation?
    b. How effective is this review meeting?
3. How good are the test phases?
    a. How many test cases are needed?
    b. How effective is a test phase?
4. What is the quality of the requirement definition?
5. What is the quality of the high-level and detailed design?
6. What is the initial quality of the code (before inspections/test)?
7. Which phase/activity has the biggest influence on quality?

This list is not exhaustive, but these are the questions that come to mind when one wants to measure and control quality. Certain questions can trigger additional questions; for instance, when it appears that a certain test phase is ineffective in finding defects, additional questions are needed to investigate the activities and their effectiveness.

Metrics:

1. Number of undetected defects in the released product
2. Number of defects found per requirement
3. Number of latent defects in a product before an inspection or a test phase (available)
4. Number of defects expected to find in an inspection
5. Actual number of defects found in an inspection (detected)
6. Number of defects expected to be found in a test phase
7. Actual number of defects found in a test phase (detected)
8. Size of the document/code
9. Detection rate: percentage of defects detected (detected/available)

The aforementioned metrics can be collected in most projects, since the data are usually already available in inspection records and defect tracking tools. But to analyze the metrics, a

measurement model is needed. Since the metrics are related, only when looking at a combination of several metrics can conclusions be drawn that help answer the questions and thus reach the goals set out for the measurements.

A Project Defect Model

The previous paragraph explained that there is need for a model to measure defect insertion and detection in order to estimate and track product quality. A pilot project was started to develop the project defect model. The project defect model is an estimation and tracking model for product quality, using the defect flow in a project. The model is based on ideas from Humphrey (1989) and Kan (2003). At the start of the project, defect estimates for all processes are made. These estimates are tracked based on data collected from the processes during the project, and the estimates and actual data are used to control the quality. The defect flow is tailored to the process that is used in a project. Figure 5 illustrates the general model that is used as a template.

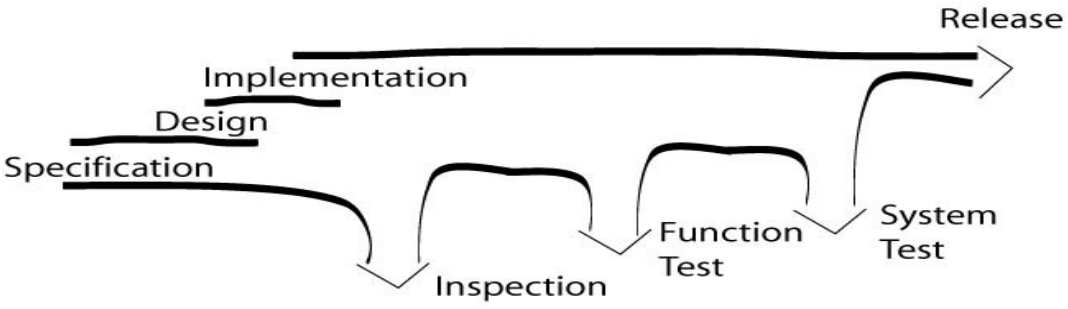

Figure 5 Defect flow

This model can be used to illustrate the goals stated previously. Goal 1 relates to the downward outgoing arrows, maximizing defect detection/reduction. Goal 2 is about minimizing the incoming arrows on the left, and goal 3 relates to the resulting outgoing arrow on the right. Goal 4 is general and relates to all arrows.

To get more insight into the quality of the product during development, it is necessary to measure the software development processes from two views: *injection* and *detection* of defects (including removal).

Injection of defects is done during the specification, design, and coding phases; defects are either injected in documents or into the actual product code. The picture shows the “inflow”

of inserted defects from the development phases. Defects are also inserted due to faulty fixes of previous defects. Measuring injection gives insight into development phase quality; this is further explained in a later section.

Detection of defects is done via inspections and test during all phases of the project. In Figure 5 the down arrows show detection. Also, the thickness of the arrow from left to right, which represents the number of latent defects in the product, is reduced. Measuring detection provides insight into the effectiveness of verification phases. Detection rate is also called *yield*:

yield (X) = (# defects removed in phase X) / ((# defects from phase X-1) + (# defects injected in phase X))

By measuring injection and detection, a project can track the number of defects in the product and determine if there are quality risks and what the origin is, for example, too many defects introduced in the product and/or insufficient testing to capture the defects before delivery to the customer. The farthest right arrow represents the number of latent defects in the product version that is delivered to the customer. This arrow should be as thin as possible.

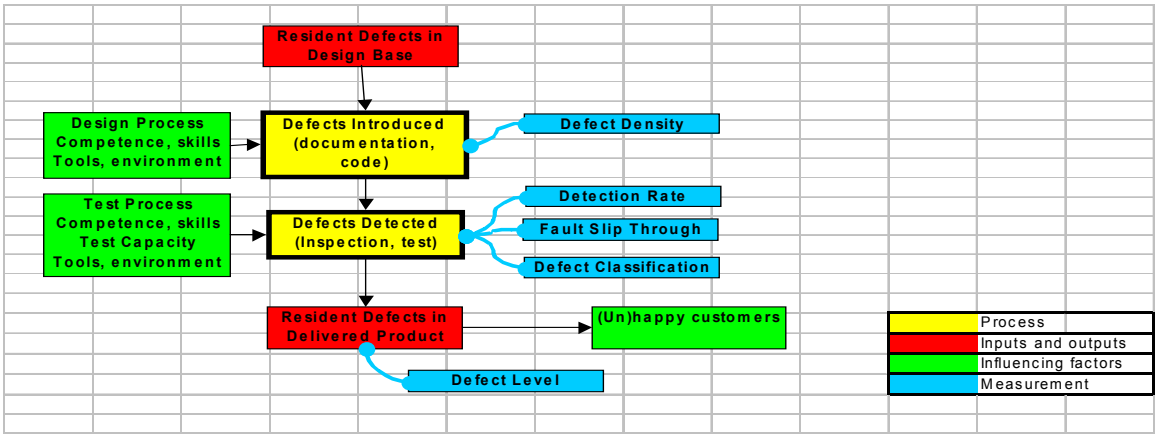

Figure 6 Process view

Several measurements are defined in the model (see figure 6):

- **Defect density**: Number of inserted defects per phase, insight on development quality
- **Detection rate**: Percent of defects detected, insight in effectiveness of verification

- **Fault slip through**: Percent and number of defects not detected in the phase they were injected
- **Defect classification**: Defect type, determination where the defect should have been found
- **Defect level**: Number of latent defects in the released product

By analyzing the measurements for one or more phases, conclusions can be drawn regarding the quality of the product. Based on these conclusions, actions can be taken to steer quality based on quality targets defined for the product. More information about the defect model can be found in (Linders 2003).

Estimating Quality

Quality predictions from the model are based on estimates of the number of defects to be made and found. Making these estimates is not easy. The best way to learn to estimate is by doing it frequently, and collecting data to validate the estimates. This is a similar process to that of estimating delivery dates or budgets; initial estimates are difficult, but after a while experience is gathered and estimates will improve.

Which factors are important for defect estimates? Following is a list of those that are the most relevant:

- Product size
- Product complexity
- Project team competence
- Project constraints
- Development environment

Estimating the number of defects expected to be injected and found can be performed in a way similar to budget or time estimates. For instance, the product to be developed can be compared with a similar product that was developed earlier; historical data from earlier projects can then serve as the basis to derive the estimates. The figures are adjusted based on the competence of the team compared to the team that developed the previous product. In this way, initial estimates are made, which can be tuned during the development of the product based on actual data.

An organization produces software with a number of defects between statistical limits based on the maturity of its development processes (see the CMMI® for more information about quantitative management (SEI 2002)). Depending on the maturity of the organization, the distance between the limits becomes smaller and the reliability of defect estimates increases. Additionally, the defect injection and detection profile across the development phases becomes repeatable, which makes it possible to estimate the number of defects in a future phase when one or more phases are finished. Based on these estimates, defect control limits can be defined. When the actual number of defects goes over a limit, it signals a potential quality risk to be investigated.

Steering on Defect Data

During the project, defect data are collected into the model. The defect database in the model comprises all defects that have been found, both from inspection and testing. Detailed data of every defect are entered, including a slogan (one-line description of the defect), references, phase detected, and phase inserted. Based on these defect data the defect flow is measured, that is, the number of defects made in a certain phase, and the number of defects found. These defect counts are compared with the estimates, and differences are calculated.

So one has the estimates on defects, and initial data in the project. But how can one actually steer the project with it? The authors apply the approach of feedback and analysis sessions, based on learning theory (see Solingen and Berghout 2001), which is very effective. Depending on the availability of data, feedback sessions are held once or twice a week. The design leaders, test leaders, and the quality engineer attend the sessions. The Excel file containing the model and all defect data are used to do online analysis, and to adjust estimates where needed. Since the sheet contains all calculations, re-estimating immediately results in adjustment of delivery and release quality levels.

The role of the quality engineer is to present the data, and ask the design and test leaders if the data match with their understanding of the project and the quality of the product. Quality engineers must be supportive in providing the data, but must also be critical during the analysis. When conclusions do not match with the data, they should not give up, but instead ask the design and test leaders how they came to their conclusion, and how they can explain that they don’t match. Often it is useful to combine several measurements to explore issues from different views. It makes it possible to dig deeper, and to get a better explanation of the

situation. Also, a quality engineer can ask more questions, thus ensuring that conclusions match with the data, and are not brought up just to have a conclusion without proper analysis.

An example on discussing data in a feedback session is the adjustment of detection estimates. If, for instance, a phase detection rate of 50 percent is expected, and 45 percent of the expected defects are detected *halfway through the phase* (that is, when 10 of the 20 planned test cases are done), then either the number of defects that is inserted is higher than initially expected, or the actual detection rate is higher – testing is more effective then expected. If the first is true, then there is a quality risk in the product since more defects have been made. Also, it gives a signal usable to improve the process phase where the defects were injected, to reduce defect injection in a next increment. However, if the estimated injection level is still valid and thus the resulting detection rate is higher, further investigation is warranted to understand how this is accomplished. That makes it possible to learn and improve verification in other projects, based on the positive experiences from this one.

The sessions stimulate discussion quality and the development approach between design and test leaders. Misconceptions become clear, which result in adjustment of priorities, more or less test cases, extra inspections, and other actions that improve quality. Since the people attending the meeting are steering the teams, they can implement decisions after the meeting, although occasionally they have to check with the project manager to see if there are implications on lead-time or budget. The data from the defect model help them to get more control, and make decisions earlier; in the end, this saves the project time and money.

EXPERIENCED BENEFITS

The model was developed and verified in a pilot project (see Linders 2003). This project was executed during 2001 and 2002. The project resulted in a prediction that the product should contain 21 latent defects, which customers would find during six months after release of the product. In reality, customers reported 20 defects. Other projects have not yet reached the end of the six-month customer period, but current estimates for those projects are also close to the initial estimates given at release of the product.

During the pilot project, the model signaled quality issues that were analyzed, the major ones being:

- Slip through of requirement defects; this was stopped after thorough architecture inspections
- Improvement of inspection meetings through moderator and inspector coaching and better planning
- Improvement of inspection preparation through improvement of checklists
- Better release decision; requirement risks where known and accepted by product management

One may question how the model supported solving such issues. Look, for example, at the release decision. Data from defects detected in phases showed that test phases discovered defects that could have been found earlier. Function test found many inspection defects, while system test discovered a lot of function test defects. Defect analysis made clear what kinds of defects were missed, and the inspection and early test processes were improved. Based on trigger analysis with orthogonal defect classification, the authors also determined test progress. Together with a requirement-based test matrix, the project predicted where requirements are sufficiently verified, and where there are risks of latent defects. Test focus and scope were improved during the project, based on data from the model, and remaining quality risks are on requirements that are seldom used. Based on the available data, product management made a solid release decision. Inflow of defects after release confirmed this decision. The product has been sold and installed at many customer sites, without any major issues, and customers are very satisfied.

In the ongoing projects there are similar benefits. Inspections improve during the project and prevent defect slip through to test, function test finds defects before shipping to system test, and process improvement and re-enforcement of design rules lowers the number of defects inserted. Final figures will be available in a future paper, which will give more insight in the sources of defects and the effectiveness of inspection and test activities from eight projects where the model has been applied.

The project defect model is beneficial for projects. The measurements help estimating, planning, and tracking quality during the projects. These quality data are used in the projects together with time and cost data, to improve decision making. The model identifies quality risks at an early stage, helping the projects taking corrective actions and decisions on product release and maintenance capacity planning. Also, the design and test teams using the model

gain significant quantitative insight into their design and test processes that is used in future projects.

CONCLUSIONS

In this article, the authors have explored a proactive approach to quality during product development. They concluded that whatever product development strategy is chosen, it is worth to pursuit a high quality level This is especially true for the early project phases. In practice, the focus on quality tends to be relatively low during product development and increases during the later testing phases. Detecting and removing defects during testing is limited and expensive. It exposes the software manufacturer to unwanted risks when releasing software products. A more proactive attitude toward managing quality is needed.

The presented project defect model allows a software project to actively manage quality from start until completion. It gives insight in the development stages where defects are introduced in the product, and it measures the effectiveness of defect detection. The presented case study shows that the model has been beneficial to projects. In particular, feedback sessions of defect data analyzed by the team proved to be very powerful.

Two closing remarks are necessary. First, applying the project defect model requires mature development processes. Very immature organizations will have no stable basis for detailed measurements, which severely hampers the process of obtaining reliable data. Furthermore, no historical data will be available to define thresholds. These organizations can only benefit from the model when they first invest in a mature development process. Second, when applying the model optimum it is of interest to explore the trade-off between appraisal cost (prerelease defect detection), rework cost (prerelease defect removal) and operational cost (post-release defect removal). This broadened view takes into account quality-related activities during the entire product life cycle, development, and operations. By extending the model in the future with cost data, it will evolve into a true implementation of a cost of quality model.

REFERENCES

Card, D. N. 1995. Is timing really everything*? IEEE Software Magazine* 12, no. 5: 9-22.

Denning. 2001. The profession of IT: Crossing the chasm. *Communications of the ACM* 44, no. 4: 21-25.

Fenton, N., and M. Neil. 1999. A critique of software defect prediction research. *IEEE Transactions on Software Engineering* 25, no. 5.

Gokhale et al. 1996. Important milestones in software reliability modelling. Communications *in Reliability, Maintainability and Serviceability, SAE International.*

Humphrey, W. S. 1989. *Managing the software process*. New York: Addison-Wesley.

Kan, S. H. 2003. *Metrics and models in software quality engineering*. New York: Addison-Wesley.

Linders, B. 2003. *Controlling product quality during development with a defect model*. European SEPG 2003, London.

Moore, G. 1995. *Crossing the chasm*. New York: Harperbusiness.

RTI. 2002.The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, Prepared by RTI for National Institute of Standards and Technology, U.S. Department of Commerce.

Sassenburg, J. A. 2002. Reviews, why and how? *Informatie* (October): 16-21 (in Dutch).

Sassenburg, J. A. 2003. When can the software be released? In *Proceedings of the European SEPG*, London.

SEI. 2002. CMMI for software engineering: Staged representation, Technical Report ESC-TR-2002-029. Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Solingen, R. V., and E. W. Berghout. 1999. *The goal/question/metric method: A practical guide for quality improvement of software development*. New York: McGraw-Hill.

Solingen, R. V., and E. W. Berghout. 2001. On Software engineering and learning theory: Facilitating learning in software quality improvement programs. In *Handbook of Software Engineering and Knowledge Engineering, volume 1*. River Edge, N.J.: World Scientific.

Wallace, D., and C. Coleman. 2001. *Hardware and software reliability: Application and improvement of software reliability models*. Software Assurance Technology Center, Report 323-08, NASA.

BIOGRAPHIES

Ben Linders is specialist of operational development & quality at Ericsson R&D, in the Netherlands (www.ericsson.nl). He has a bachelor's degree in computer science, and has done a master's study on organizational change. His focus is on implementing high-maturity practices, with the aim of improving organizational performance, bringing business benefits. Since 2000 he has lead the Defect Prevention program. He has defined and applied a project defect model, used for quantitative management of the quality of products and effectiveness of verification. He can be reached by e-mail info@benlinders.com.

Hans Sassenburg received a master's of science degree in electrical engineering from the Eindhoven University of Technology in 1986 (The Netherlands). He worked as an independent consultant until 1996, when he cofounded a consulting and training firm (Alert Automation Services b.v.). From 1996 until 2001 he worked as a guest lecturer and assistant professor at the Eindhoven University of Technology. In 2001 he moved to Switzerland where he founded the consulting firm SE-CURE AG, (*www.se-cure.ch*), offering services in the field of applied business/software metrics. Having finished his first book, he started his doctoral research at the Faculty of Economics at the University of Groningen (The Netherlands) designing a model to support release decision making for strategic software applications. He can be reached at +41 33 733 4682, or by e-mail *hsassenburg@se-cure.ch*.