Towards Proactive Management of Technical Debt by Software Metrics

Anna Sandberg[1)], Miroslaw Staron[2)], Vard Antinyan[2]

[1]*Ericsson*
Goteborg, Sweden
anna.sandberg@ericsson.com
[2]*University of Gothenburg*
Hörselgången 11, Göteborg, Sweden
vard.antinyan@cse,gu.se, miroslaw.staron@cse.gu.se

Abstract. Large software development organizations put enormous amount of effort not only for responding to continuous requests of customers but also for reengineering and refactoring activities to keep their product maintainable. Often rapid and immature feature deliveries over long period of time gradually decrease the product quality, and therefore the refactoring activities become costly and effort-intensive. This situation is described by the concept of "technical debt", which represents the accumulated rework that organization has to do in order to prevent the slowdown of the development. In this paper we report results of a case study at Ericsson on using software metrics for moving towards proactive management of technical debt. Our observations show that there are four distinguishable maturity phases of quality management over the eight years of development time of two large products: Start-n-stop, Reactive, Systematic, and Proactive quality management. Three sophisticated metrics are applied to help the organizations to move towards Proactive management of technical debt. These metrics are used on a systematic basis to provide information on the areas of the product that have tendency of accumulating technical debt. Software engineers use this information for making decisions on whether or not the pinpointed areas should be refactored.

Keywords: Software development, Software metrics, Software technical debt

1 Introduction

Large software developing organizations want to spend their time on feature development and innovations to be competitive and profitable. In practice, far too many instead spend a substantial part of their time on managing non-feature-delivering activities, such as defect handling, refactoring, and scope-cutting. This inefficient situation can be explained with help of "technical debt", which is a known metaphor [1], but has still gained too little practical attention in the software development industry. Gradually accumulated technical debt in a long period of time can reach to scales that

require enormous effort for its control and management. Furthermore, it obliges organizations to stop feature development activities from time to time and focus only on defect handling and refactoring. This kind of development is labeled at Ericsson as Start-n-stop development with Start-n-stop quality management. In contrast with Start-n-stop development, every organization aims to move towards Proactive development, described by proactive management of technical debt in parallel with continuous feature delivery. Continuous delivery allows delivering new features to the customers all the time, thus making the organization competitive in the market [2] [3]. However, gradual increments of technical debt over long period of time slow down the development process. This situation requires continuous quality management as well which permit continuous control of technical debt. Approaches for technical debt management exist as well [4, 5]. However there is scarce data reported on continuous technical debt management, which ultimately permits Proactive development. The research question we raise in this paper is:

> *How can we use software metrics to proactively manage technical debt in a large software development organization?*

The telecom company Ericsson has since many years worked with metrics to overcome the challenges around technical debt. In this article we have documented these experiences to visualize and present how large software development organizations can increase their understanding and manage the technical debt in four different maturity phases: Start-n-stop development, Reactive development, Systematic development and Proactive development. Proactive development is the want-to-be phase with its capability of proactive quality management and comparably high attention to feature development. We show how three sophisticated metrics can guide software organizations to proactively manage technical debt with the goal of spending valuable development time on feature-delivering activities and innovations.

2 Technical Debt Challenges

The price to pay for software development includes more than new features with added functionality or increased performance. Too often, poor quality, stemming out from rapid immature feature delivery, requires costly reworks during later software releases. In other words, if a customer has $100 to spend, she cannot buy features for all of it. She has to spend a part of it on architectural features and defect handling. The size of this part is dependent on the size she spends in the previous releases. The challenge is to accept that fact and include long-term value thinking when managing her ongoing software development. A similar and understandable metaphor is linked to paying interests [1]. If she lends $100 to zero % the first quarter, it is easy to understand that she will need to pay much more in the upcoming quarters.

Krutchen et al. [6] has divided the software development content to four categories based on the (non-) feature-delivering (i.e. negative and positive value) and visibility (see Figure 1). The guiding principle is – *what is possible to see is possible to manage*, especially when bringing a positive value. All software organizations tend to pay

attention to features and defects just because they are visible. It is the invisibility and lack of tangible customer value, which characterizes the *technical debt* that makes it harder for the software organizations to manage it. Therefore, it is extremely important to measure and visualize how technical debt can grow over time and latterly eat up the capacity for feature delivery if not kept under control.

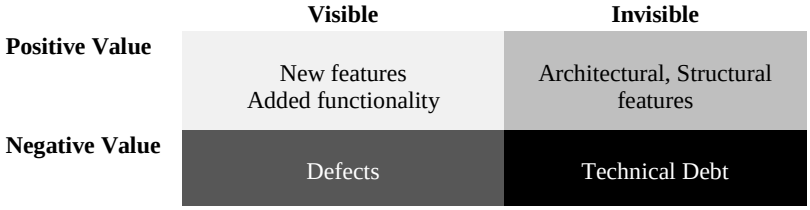

Figure 1 Technical Debt as an invisible negative value (Kruchten et al. 2012).

Over time Ericsson has used a variety of software metrics to manage different aspects of technical debt [7-9]. We investigated how software metrics are used in the studied organization which helped the organization move from Start-n-stop development to Proactive development.

3 Research Approach

The studied Ericsson case consists of two different products, each containing several millions lines of code developed over a 20 years period. Both products are developed in global multi-site environments with development sites in three different continents. Each development organization has ~500 developers. The general interest in in software metrics at Ericsson has been high during a longer period [10], which has driven the interest of the organization to dive into challenging, but beneficial software metrics areas such as technical debt.

We studied metrics used by the two products over an eight years period. We used triangulation of multiple data sources (document analysis, observations and literature studies) in several iterations, so in the end we could identify four typical development phases with their corresponding quality management practices. We call them Start-n-stop development (correspondingly with Start-n-stop quality management), Reactive development, Systematic development and Proactive development. It is the Proactive development phase, in which the organization can invest the majority of its capacity on feature-delivering work and at the same time proactively manage technical debt. We studied the use of metrics and their evolution over time in the organization, which provided insights on how the best metrics were chosen for practical use. We also identified how metrics can be applied systematically and distributed across the organization, among smaller substituent development branches, which helped the organization to transform their quality management practices toward Proactive quality management.

4 Applied Metrics

Over time, Ericsson has used variety of metrics in different manners and for different purposes. With the development maturity the use of metrics and their role has evolved considerably in the organization. In the coming subsections we present the technical debt metrics and their use in the four distinguished maturity phases.

4.1 Using Metrics in Start-n-stop Quality Management

As in every large software development organization, Ericsson also measured and still measures such aspects as software defects, test coverage, size, velocity etc. Such elementary measurements were used long time in the company and are vital for decision making. In the first maturity level (Start-n-Stop) the organization only focuses on visible aspect of quality management. Namely, the organization can only see the software defects as a manifestation of software quality. Therefore, developers mainly focus on handling defects to improve the quality. In this maturity phase the organization had either very scarce use of technical debt metrics or did not have any use at all. As the product grows, the size and complexity of the product escalates, and therefore technical debt accumulates. This situation prompts developers to start irregular measurements (usually taken place right before the product release) and localized refactoring activities. This is the beginning of transition to Reactive quality management.

4.2 Using Metrics in Reactive Quality Management

As the development gains more maturity, among all aforementioned metrics the organization also uses metrics which provide insights on invisible aspects of quality (technical debt). All these metrics are summarized in Table 1.

Table 2 Early technical debt metrics used at Ericsson

Metric	Used to measure Tech. Debt of
Number of added LOC	Code
Number of deleted LOC	Code
Number of modified LOC	Code, tests
Number of developers	Code
Fan-in	Code, architecture
Fan-out	Code, architecture
Cyclomatic complexity	Code
Nesting	Code
Halstead measures	Code

The application of these metrics usually corresponds to Reactive quality management, when the organization needs to have an insight about the quality of their product just before the delivery (and therefore before getting defect reports of customers). The measures are usually applied in ad-hoc manner and by isolated individuals or

teams. The measures are not used by combination and with determined thresholds. Rather raw numbers of them is presented and the rest of the verdict is left to the observing developers. In such situations, usually small areas of the product turn out having obvious technical debt, and the organization makes decision on either taking the risk or refactoring the identified areas.

4.3 Using Metrics in Systematic Quality Management

In the phase of systematic quality management the emphasis is put not only on what metrics are used but on how they are used. In this phase a combination of several simple metrics are used to achieve a single more insightful indicator. Powerful visualization techniques are used for visualizing big data for organization. Most importantly, in order to monitor the quality of the product systematically and on all organizational levels, measurement systems are developed which can provide information on weekly or daily basis [11]. The value of the metrics is enhanced when they are visualized in appealing and simple diagrams.

Figure 2 and Figure 3 show three key metrics for systematic assessment of technical debt at Ericsson. The metrics are developed in an integrated dashboard which serves the whole development organization. Together they form a solid metrics system, which then can guide the software development to continuously build in quality from the start. More importantly, the solid metrics system increase awareness so that the organization immediately acts on problems to always secure high quality and a continuously improved software development base.

The source code stability metric (heat maps) are used to visualize the change frequency of code [12] (see Figure 2 left hand diagram). This is a comprehensive way of showing thousands of software changes in the code in a single figure. The dark areas draw the attention to and trigger discussions about the effectiveness of testing of these areas and also other negative development practices.

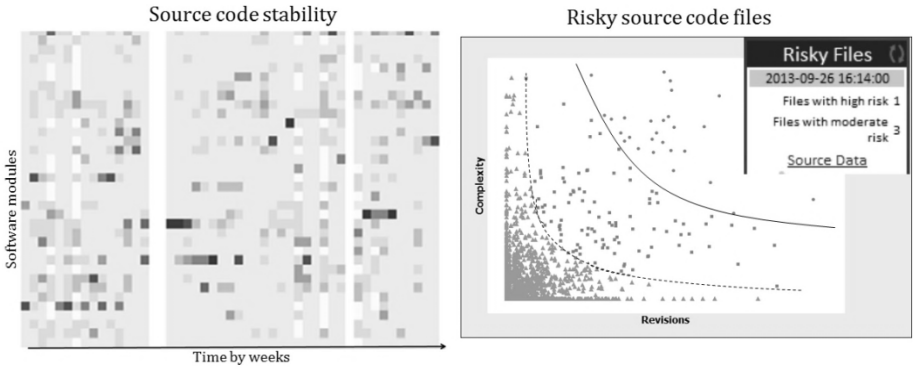

Figure 2 Key metrics and their visualization: Code stability heat map and dashboard of risky code

Studying the repetitive patterns in the heat maps allows predicting the changes, informing designers about potential need to update components and directing testing to reduce the risk of undiscovered defects.

Visualizing risky source code files [13] (see Figure 2 right hand diagram) permits developers to identify the areas of source code that are defect prone or difficult to maintain. Two metrics are combined for this assessment: cyclomatic complexity and number of revisions per file. Two thresholds are distinguished for this measure in order to separate files with high risk (upper right cloud of dots) and files with moderate risk (dots in between the two thresholds). In the upper right corner of the diagram the information product is presented, which shows the number of files with high risk and the number of files with moderate risk. The information product is integrated in the metrics dashboard of the organization alongside with other important metrics (not necessarily metrics concerned with technical debt). Files with high risk are refactored or additionally tested by all means. Depending on the organization's resources and risk appetite they can decide whether or not moderate risk should be mitigated or not.

Visualizing implicit dependencies [14] (see Figure 3) brings the invisible to become visible and thus draws attention to actions which shrinks the area of technical debt. When using historical data to predict events, the organization can reduce the negative repetitive patterns with help of for instance different preparation techniques.

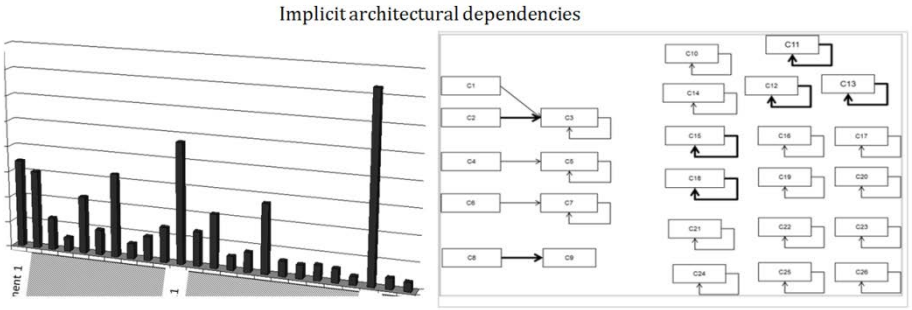

Figure 3 Key metrics and their visualization: Implicit architectural dependencies

The left side of the figure is the graphical representation of change waves, which allow detecting implicitly interconnected software modules (files in case of Ericsson). This is done by detecting how frequent changes in one file trigger frequent changes in another file after certain development time. In that diagram we can detect such patterns by observing different change-frequency picks. Based on this information the right hand diagram is developed which shows the names of modules and their implicit dependencies. By reviewing these dependencies the design architects make decisions on reengineering and avoiding unwanted dependencies.

4.4 Towards Using Metrics in Proactive Quality Management

Ericsson currently makes transition towards using metrics for proactive quality management. At the time of writing this paper the organization was adopting a princi-

ple of using a standard set of metrics for independent development teams and their development branches. The idea is that every team should have the possibility of interactively using the three metrics when developing the code. This permits to get the earliest possible feedback on their code quality and immediately take action if necessary. We also observed that with this kind of quality management a challenge emerges concerned with knowledge of developers on interpreting the metrics values. In order to have quality control of this level, the developers need to know what exactly the metrics' values show, and how they can use them on their own for improving the quality.

5 Action Principles for Staying in The Phase of Proactive Quality Management

Ericsson has longitudinal experiences of successful software improvement activities [15]. Their efforts to understand and manage technical debt come therefore rather natural. However, the organizational will to produce customer-value is also natural and understandable as adding no "customer-value" is equal to "no business". When this will expands as a result of grasping over too many appetizing business opportunities, the outcome is a decreased feature-delivering capacity. The will needs to be accompanied by deep understanding of software development and its invisible technical debt. The experiences at Ericsson suggest the following action principles for moving towards and staying current in the optimal phase of Proactive quality management:

- **Embrace Technical Debt Existence**. Realize that all software development comes with a price for managing technical debt. Make use of metaphors and visualizations of metrics to create organizational acceptance of the term Technical debt. "Lending money to zero % interest… - the first quarter" is difficult to misinterpret. Seeing how the feature delivering capacity increases is the most inspiring to ensure action.
- **Understand Technical Debt.** Understand the deeply underlying factors for technical debt appearance. Continuously analyze the product and visualize metrics in all organizational levels in the earliest development phases. This reveals details about how neglecting design problems can turn into stinker-code over time. Understanding such cases allows the organization to plan and prioritize accordingly to avoid them and by that reduce their negative effect.
- **Start with the obvious**. Start with the small and most crucial set of problems. Usually the most of the problems in the product are concentrated in the few of artifacts. Use already identified and managed portions for understanding knowledge accumulating for the next step of action.
- **Learn and improve based on maturity phase**. Understand the maturity phase that is unique for your organization and by that implement and make use of the metrics most beneficial in that context.
- **Gradually increase towards more product-oriented metrics**. When the obvious metrics are in place, start with in-depth product oriented metrics like

complexity or implicit architectural dependencies in our case. These metrics prepare the organization to uncover the hidden technical debt in the product.

- **Provide Solid Metrics Systems.** Triangulate metrics to understand high complexity products in-depth. It is important to provide solid and visual metrics systems from which root-cause-analyses can be done. Appealing and easily accessible metrics facilitate the root-causes activities further. Combining the internal properties' metrics together allows keeping track of the influence of the technical debt on product performance.
- **Experiment with New Metrics.** Experiment with new metrics to find the most suitable and applicable ones. Experiences show that there are good practical metrics to re-use, but every organizations act in its own unique context, where different practical metrics can make tangible difference.
- **Do not stop paying attention.** Once in the want-to-be Proactive development phase, it is not equal to staying current there. Continuous attention to always act on trend changes is of highest importance. When allowing trend slippage, the organization allows the technical debt to grow and capacity for feature-delivering activities to shrink.

The notion of technical debt allows organizations to reason about the need to increase quality and organizing the road to understand and manage the technical debt. The four maturity phases provide a roadmap and improvement opportunity for large companies to manage the technical debt in practice. By using a solid metrics system, organizations can continuously monitor and act on negative trend deviations. When doing so, they can get the most out of the important feature-delivering activities – value delivery to customers and innovation!

6 The Journey Towards Proactive Quality Management

Along the evolution of organizational awareness of the invisible aspects of development (see Figure 1), the proportions of effort spent on each of these aspects change (see Figure 4). The effort spent on these aspects does not change by default along with increased awareness. It is the awareness itself that increases the will as well as a sense of urgency to act in ways where much attention is directed to manage the invisible aspects. Figure 4 visualizes the roughly estimated proportion of development effort spent on each of the four elements and through four maturity phases. Moving forward, we describe each maturity phase and typical practical metrics in more detail.

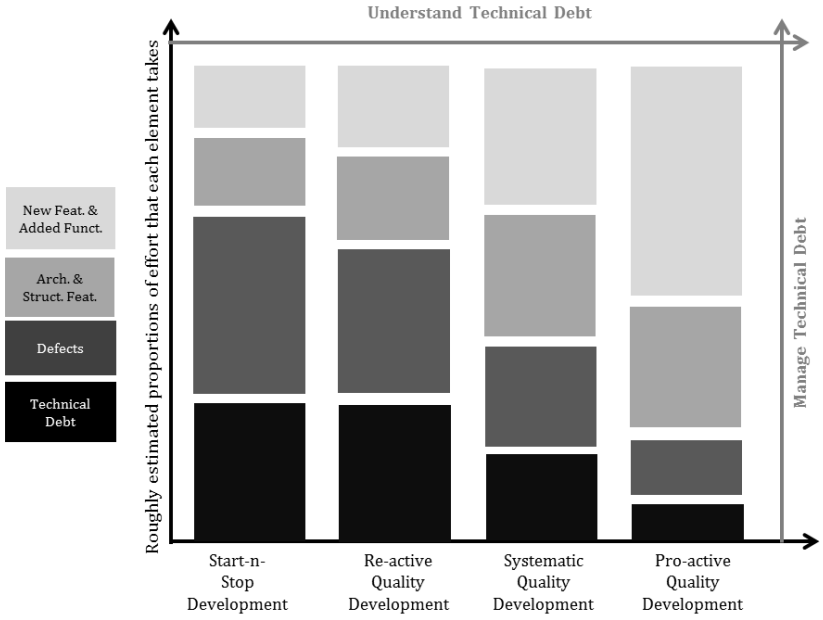

Figure 4. Estimated proportion of development effort per element in each maturity level

6.1 Start-n-Stop Quality Development

In the Start-n-stop development phase the organization typically focuses on feature development (i.e. the visible positive value) and then reacts on defects when the quality situation becomes critical. The technical debt is not managed at all and remains "hidden" until the development of the product decelerates to the extent when the organization needs to react immediately. This deceleration is usually observed by a high number of in-development defects which need to be fixed before progressing with the development. The organization "stops" new functionality development and focused on reduction of defects only [16].

The main metric focus of the organization in the start-n-stop development phase is the number of known defects measured from various perspectives (e.g. per severity, release, function, etc), which are perceived as the major problem. Defect volume metrics guide the development and the key mechanism used is to stop development when high defect levels prevent continued development. During such a stop, a 'clean-up' period is initiated before the development can start again. In this phase the technical debt is hidden and only the symptoms, defects of it, are managed. To continue the journey, the organization needs to develop fundamental awareness of its inefficient development practice.

6.2 Reactive Quality Development

Once the organization recognizes the need to make in-depth improvements it usually seeks methods to reduce the amplitude between the high and low number of defects. Using complexity analyses is one beneficial method as it is perceived to provide the organization with understanding of the underlying problems that cause the defects (i.e. allowing the organization to start managing the invisible values) [17]. In this phase, the organization starts to reactively act on its situation and typically improves its architecture management. Common efforts are focused on refactoring of stinker-code and redesign of interfaces. These improvements result in lower number of defects and thus more capacity for feature development. In this phase, the organization has a limited understanding of the technical debt although there is a shared perception that defects are only symptoms.

A few senior developers are constantly prioritized to do some localized discrete measurements and improvements here and there in the product to keep the development ongoing. Metrics in this phase draw the attention of the organization to quality assurance and product properties rather than quality problems. Typically used metrics are test effectiveness, test-requirement coverage, basic complexity measurements and product availability. We also discovered that software designers make ad-hoc attempts to visualize various aspects of product stability. Visualizing instability in this context is a means of searching for the areas where technical debt can be found.

In this phase the technical debt is recognized and attempts are made to understand it. To continue the journey, the organization needs to develop in-depth understanding of its actual problems that are explaining their current development situation.

6.3 Systematic Quality Development

In this phase the organization understands the need for managing the technical debt and has established measurement systems and visualization tools. The organization attempts to have a standard shared metrics dashboard or similar tools so all development teams, developers and architects can follow the evolving product condition. The main focus is to find new means for visualizing the invisible aspects of the product to be able to manage them. The organization spends effort on defining new metrics for measuring the invisible problems in order to remedy them before they even occur. We observed this type of behavior by studying architecture quality [14].

The metrics for quantifying the invisible values are organized in a solid metrics system. This metrics system guides the development to continuously act on potential problems which can become defects (e.g. monitoring implicit architectural dependencies and combination of several complexity metrics). Stability of the source code is also monitored using code change rate metrics to identify periods of development when too intensive development prevent quality assurance from being effective. An outspoken strategy to build quality from the start is growing and selected individuals lead by example.

In this phase the complete technical debt is understood and valuable efforts are made to manage it. Practical and solid metrics systems are used to guide the organiza-

tion. To continue the journey, the organization needs to develop skills to continuously and immediately act on the metrics describing their development situation.

6.4 Proactive Quality Development

In the Proactive quality management phase the organization possesses data and experience on which aspects of technical debt can be continuously monitored. The organization has an established metrics system to both understand and manage the technical debt and can focus the main part of their development capacity on feature development. The solid metrics system monitors deviations from the decreasing trend in technical debt, thus allowing the organization to come in control of their debt at all times.

A solid standardized (in the scope of organization) measurement system continuously provides information to the architects, managers, and technical leaders. Besides this every development team and individual developer has the standardized and approved measurement tool on his own computer in order to interactively follow his and teammates' code quality and control it in its development earliest phase. The designers of measurement systems and metrics (researchers and responsible engineers from the organization) set up systematic presentations and training session, so the developers can understand the meaning and interpret the metrics.

In this phase the technical debt is both understood and managed. Focus can be fully directed to feature growth and metrics are used to automatically keep the technical debt under control. To stay current in this phase, the organization needs to continuously pay attention to metrics showing negative trend changes and immediately address this with appropriate actions.

7 Related Work

An interesting discussion on technical debt is provided by Buschmann [18] who discuss the trade-off between paying or not paying accumulated technical debt. He claims that technical debt is similar to financial debt as it accumulates like a compound interest, but it is not always paid back if the organization decides to obsolete their old product and start with a completely new one. Lim, et al. [19] observes that measuring technical debt is a difficult task as it can have a variety of manifestations. Possibly that is the reason that many organizations including Ericsson strive for establishing the right metrics system for technical debt measurement. Tom, et al. [20] explore the causes of technical debt and found that it is mainly the decisions of non-technical people that prompt accumulating technical debt. Martini, et al. [21] studies the technical debt issues in large software development companies and concludes that the lack of knowledge is not a primary cause for accumulated technical debt. However, he founds that schedule pressure for feature delivery, using legacy systems, and small time allocated for refactoring are the main causes. In a later study Martini, et al. [22] develop a qualitative model for understanding technical debt in large software development projects. This study is also conducted in the same organization of Erics-

son as our study. Our study can be considered a complement to their study as we emphasize the use of measurement system for technical debt detection. Using such measurement system with their qualitative product specific models can be a powerful tool for technical debt management.

There have been a few studies proposing approaches for identifying one or another manifestation of technical debt: Marinescu [23] propose an approach for finding the technical debt of design flows based on eight types of flows found in code. Then he investigates how different software attributes, such as complexity and cohesion, influence each of the defined design flaw. The measurement systems proposed by us for technical debt management can be used to find several design flaws proposed by Marinescu. Nugroho, et al. [24] introduces a "return on investment" approach for managing technical debt. They estimate the maintainability of system by measuring several attributes of code and categorize the units of code by three different levels of risk. Then they calculate (to unclear precision) the return on investment for code maintainability in case the risky source code is refactored and improved to a level of optimal design. The study is interesting as it explicitly attempts to develop a connection between internal quality and business decisions in software development. Guo, et al. [25] explores the effect of technical debt in practice and observes that it has significant negative influence on the studied developed project. They conclude that business factors should be incorporated in the technical debt management model so the trade-offs between business opportunities and software quality can be considered. Probably one of the best known approaches to manage technical debt is proposed by Letouzey [26]. The approach is based on complexity measurement, dependencies, and coding violations. There is also a tool support for this approach (SonarQube tool). The tool offers effective visualizations, which can come to handy for large software products. It also attempts to derive the effort required for paying the technical debt, but this is not yet rigorously tested. The SonarQube has large amount of predefined rules for detecting coding violations and can categorize the severity of violations. However the tool relies on rather simplistic measures which are shown in literature not to be so good indicators of internal quality. We believe that if the tool could rely on more sophisticated measures (combination of measures), the assessment accuracy of SonarQube would increase significantly. We show such an attempt of using sophisticated measures at Ericsson, however such measures should be further evaluated rigorously which is the future work of this research.

Tom, et al. [20] investigates the main areas of technical debt and propose a framework for its management. This work is particularly valuable due to its extensive elaboration on different kinds of technical debts.

8 Conclusions

The primary aim of software development companies is to deliver value to their customer and be innovative. However they do not spend all of their resources for value delivering activities directly. In fact much time is spent on such activities as defect handling, re-architecting, and refactoring. A relevant metaphor to describe this

situation is the “technical debt”, which can be considered the time or effort that the organization should pay in order to improve the degrading quality of the product. This paper distinguishes four maturity levels of managing technical debt at Ericsson: Start-n-stop, Reactive, Systematic, and Proactive management. We observed that Ericsson has been using metrics differently through the four maturity phases. We investigated and presented the metrics and their evolution over four maturity phases of development. The investigations showed that there are three key metrics which are used for Systematic technical debt management at Ericsson: 1) change frequencies of files visualized by heat maps 2) implicit architectural dependencies and 3) risky source files visualized and reported daily on a dashboard. We also observed that in order for the organization to move towards Proactive management of quality and technical debt in particular, every software development team and individual developer is preferred to have a standard and organizationally accepted measurement dashboard on her own computer, so she can interactively enhance the quality of the product all the time.

Acknowledgement

The research has been conducted in Software Center, Chalmers | University of Gothenburg. http://www.software-center.se/
The researchers thank the manager of Research and Development organization at Ericsson, as well as all design architects and developers who supported the research.

References

1. W. Cunningham, "The WyCash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, pp. 29-30, 1993.
2. P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*: Pearson Education, 2007.
3. J. Bosch, *Continuous Software Engineering*: Springer, 2014.
4. Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 31-34.
5. N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 47-52.
6. P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: from metaphor to theory and practice," *IEEE Software*, pp. 18-21, 2012.
7. V. Antinyan, M. Staron, J. Hansson, W. Meding, P. Osterström, and A. Henriksson, "Monitoring Evolution of Code Complexity and Magnitude of Changes," *Acta Cybernetica*, vol. 21, pp. 367-382, 2014.
8. N. Ohlsson, M. Helander, and C. Wohlin, "Quality improvement by identification of fault-prone modules using software design metrics," in *Proceedings: International Conference on Software Quality*, 1996, pp. 1-13.

9. K. Pandazo, A. Shollo, M. Staron, and W. Meding, "Presenting software metrics indicators: a case study," in *Proceedings of MENSURA 20th International Conference on Software Product and Process Measurement, vol. 20, no. 1*, 2010.
10. A. B. Sandberg, L. Pareto, and T. Arts, "Agile collaborative research: Action principles for industry-academia collaboration," *Software, IEEE,* vol. 28, pp. 74-83, 2011.
11. I. I. 15939:2001, "Information technology — Software engineering — Software measurement process," 2001.
12. M. Staron, J. Hansson, R. Feldt, W. Meding, A. Henriksson, S. Nilsson, *et al.*, "Measuring and Visualizing Code Stability--A Case Study at Three Companies," in *Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on*, 2013, pp. 191-200.
13. V. Antinyan, M. Staron, W. Meding, P. Osterstrom, E. Wikstrom, J. Wranker, *et al.*, "Identifying risky areas of software code in Agile/Lean software development: An industrial experience report," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, 2014, pp. 154-163.
14. M. Staron, W. Meding, C. Hoglund, P.-E. Eriksson, J. Nilsson, and J. Hansson, "Identifying Implicit Architectural Dependencies Using Measures of Source Code Change Waves," in *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, 2013, pp. 325-332.
15. A. Börjesson and L. Mathiassen, "Successful process implementation," *Software, IEEE,* vol. 21, pp. 36-44, 2004.
16. M. Staron, W. Meding, and B. Söderqvist, "A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation," *Information and Software Technology,* vol. 52, pp. 1069-1079, 2010.
17. B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili,* vol. 426, 2005.
18. F. Buschmann, "To pay or not to pay technical debt," *Software, IEEE,* vol. 28, pp. 29-31, 2011.
19. E. Lim, N. Taksande, and C. Seaman, "A balancing act: what software practitioners have to say about technical debt," *Software, IEEE,* vol. 29, pp. 22-27, 2012.
20. E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software,* vol. 86, pp. 1498-1516, 2013.
21. A. Martini, J. Bosch, and M. Chaudron, "Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study," *Information and Software Technology,* 2015.
22. A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, 2014, pp. 85-92.
23. R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development,* vol. 56, pp. 9: 1-9: 13, 2012.
24. A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 1-8.

25. Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. Da Silva, *et al.*, "Tracking technical debt—An exploratory case study," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 2011, pp. 528-531.
26. J.-L. Letouzey, "The SQALE method for evaluating technical debt," in *Proceedings of the Third International Workshop on Managing Technical Debt*, 2012, pp. 31-36.