Dashboards for continuous monitoring of quality for software product under development

Authors:

Miroslaw Staron[1)], Wilhelm Meding[2)], Jörgen Hansson[3)], Christoffer Höglund[4)], Kent Niesel[5)], Vilhelm Bergmann[4)]

[1)] University of Gothenburg, [2)] Ericsson AB, [3)] Chalmers University of Technology, [4)] Saab Electronic Defense Systems, [5)] Volvo Car Corporation

1 Introduction

Qualities and architecture of modern software products are distinguishing excellent products from the average ones and can influence the success of software development companies. Extensible and scalable architectures of products combined with Agile and Lean software development can determine a long-term sustainable businesses and maximize the value which customers get from these products. However, Agile and Lean software development methodologies come with a set of new challenges for monitoring and managing software quality in large software products (Tomaszewski et al., 2007). Continuous deliveries of customer value (features) need to be supported by extensible, reliable and well-designed software architecture which in consequence demands shorter feedback loops on quality of the product and its architecture before and after release. Combining the extensible and sustainable software architecture able to bare long-term evolution of complex software products and at the same time enabling agility can be exemplified by software products which are heavily affected by the evolution from traditional to Agile and Lean software development are telecom nodes with millions of lines of executable code, embedded software in cars with millions of lines of code or defense products with hundreds of thousands of lines of secure dependable code. In these products, monitoring quality needs to be done in two facets – number of defects for upper management complemented with details on quality attributes (e.g. reliability or performance) and agility of the software development organization.

The combination of extensible architectures, size of the product and rapid feature development cause new information needs to emerge (Buse and Zimmermann, 2012) – monitoring whether the desired quality level is achieved and when it is achieved, monitoring that it stays at that level. These new challenges require also new ways of thinking when designing and building measurement systems visualizing large number of measures in form of dashboards (Staron et al., 2008) with two facets – for upper management and for teams. The dashboards are needed to collect the necessary information in form of indicators and measures, visualize it in a simple way and spread the information about the status of indicators within the organization.

In this chapter we use the term *dashboard* to refer to a measurement system visualizing a small number of indicators (5-9) complemented with larger number of base and derived measures (>10). The dashboards contain visualization components, calculations of indicators, base and derived measures and can be in form of MS Sidebar gadgets, web pages/portals or Excel files with visualizations. They are available for upper management to control the company and for self-organized software development teams to monitor and communicate the status of the quality of their product under development (Sharp et al., 2009). Additional goals for the dashboards are for the teams to monitor of software econometrics (Ward-Dutton, 2011), monitor the progress of verification and validation or visualize architectural dependencies in software products (Sangal et al., 2005).

This chapter contributes with a systematic overview of good examples of how dashboards are used to monitor quality of software products under development – both using multiple measures and a single indicator which combines quality and development progress. In this chapter we extract recommendations for building such dashboards for practitioners by exploring how three companies use dashboards for monitoring and controlling external and internal quality of large software products under development. The dashboards presented by each company contain a number of indicators each and have different premises due to the domain of the product, its purpose and the organization. We describe a number of common principles behind a set of measures which address the challenge of quantifying readiness to deliver of software products to their end customers. The experiences presented in this chapter come from multiple case studies at Ericsson, two studies at Volvo Car Corporation (VCC) and one at Saab Electronic Defense Systems in Sweden. All companies have a long experience with software development and have undergone a transition into Agile and Lean software development; however the experience with these new paradigms differs from two to five years depending on the company. The difference in the experience provide a possibility to observe that companies with longer experience tend to focus on using measures to support self-organized teams whereas companies with shorter experience tend to focus on using measures to communicate the status from teams to management.

The experiences presented in this chapter address the following challenges of successful dashboards for monitoring quality of products under development:

- *How to standardize the measures and indicators for monitoring development progress taking into account product quality rather than schedule?*
- *What kind of measures and indicators should be used in dashboards to monitor the development progress?*
- *How to combine the need for indicators attracting attention of management when needed, with the need to collect larger number of measures for drill-down?*
- *How to visualize measures in order to effectively trigger decision processes?*
- *How to assure high quality of information provided by the measurement systems?*

The results show that there dashboards can be organized either as a single indicator (release readiness), a set of measures of development progress (e.g. number of model revisions) or as a combination of internal quality and external quality (e.g. complexity and number of defects). Depending on the presentation method the dashboard trigger different kind of decisions at each studied company.

This chapter is structured as follows. Section 2 describes how the development of large software products is organized according to Agile and Lean principles at the studied companies. Section 3 discusses elements of successful dashboards which visualize the release readiness and software development quality progress. Section 4 presents how the three studied companies realized this concept, complemented with recommendations for other companies in section 5. Section 6 provides a number of useful directions where interested readers can find more information. Section 7 contains the conclusions.

2 Developing large software products using Agile and Lean principles

In software engineering in general, release management is based on the delicate balance between the market pull for new features, and the technology push and company's strategy (Phaal et al., 2004). One of the main activities of release management is naturally the release planning activity which is concerned with which features should be implemented in the product and when they should be delivered to the end customers (Ruhe, 2003, Ruhe and Saliu, 2005).

In modern Agile and Lean software practices (Poppendieck and Poppendieck, 2007), once the features are planned, their development is usually in the hands of self-organized Agile software development teams, which use the products' architecture to design, implement and test new product features. The self-organized teams are responsible for detailing the functional (FR) and non-functional requirements (NFR) for the feature, planning of feature development (PM), systemization and feature design (SM), implementation and detailed design (DM) and initial testing (usually functional testing) (Staron et al., 2010b, Tomaszewski et al., 2007) – all with the goal to deliver a fully functional feature to the main code branch as depicted in Figure 1.

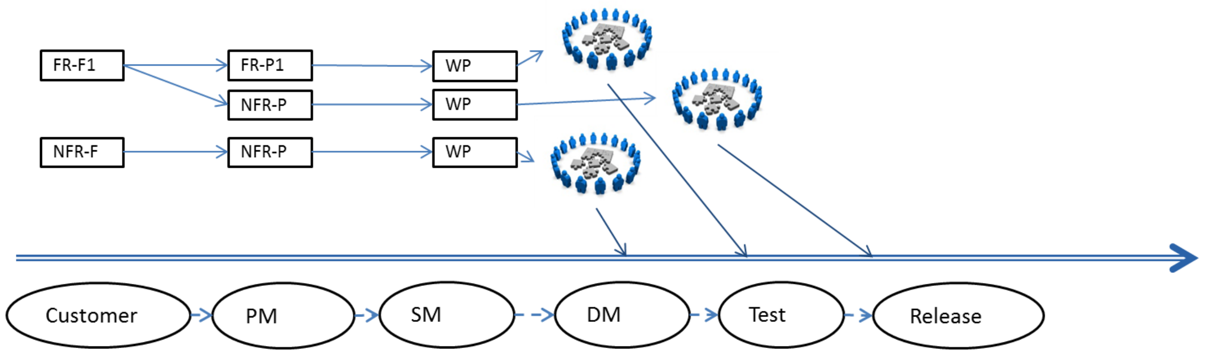


Figure 1. Conceptual view of software development flow in organizations with multiple, parallel cross-functional teams

In order to keep in pace with the market demands there are multiple teams who deliver their fully developed functional feature to the main code branch, where the feature is tested in the context of the real hardware and later in the context of the real execution environment – e.g. telecom networks, complete vehicles or defense field testing.

Monitoring the *agility* can be challenging as it comprises both the functional deliveries and quality of the product (both its features and its architecture). Release plans for features are developed based on experience and market needs. The plans are monitored so that the software development organizations are confident that the plans hold. Using indicators the whole software development program (a set of self-organized development teams working on the common code base) can monitor the validity of the plan and address the issue of how confident the development program is that the plan holds. This monitoring of agility can be in the form of *release readiness* indicator – an indicator which shows when a particular set of features is ready to be released. The indicator varies from company to company but the principles of how it is defined, calculated and visualized are the same. The prerequisite for this indicator is that the software architecture has a good quality.

In this chapter we use the ISO 9000:2005 (ISO, 2005a) definition of quality, i.e. degree to which a set of inherent characteristics fulfills requirements. We also distinguish between the internal and external quality as defined in ISO 25000 (ISO, 2005b) and use the external view of in the remaining of the chapter except for section 4.3 where we present both internal and external quality in the studied dashboards.

Monitoring the development progress is based on identifying software development phases at the company. The phases reflect how the development is organized and help to identify indicators for monitoring the progress of development of the features by monitoring the indicators. Most software development processes usually include such high level phases as requirements breakdown, software design or modeling and testing. Examples of weekly measures for those phases are:

- # of requirements in requirements' backlog for the requirements phase
- # of model/code check-ins for the development phase
- # of violations of architectural rules in models/code

- % of passed test cases for the test phase

The measures support the teams in communicating the status of the development and are succinct enough to be used in communication upwards toward project leaders and management teams. As we explore in section 4 these types of measures are simple enough and together with other components of dashboards are enough for the dashboard to be successful.

The measures of test progress include tests of non-functional properties of the architecture and the product, in particular performance, reliability, stability, availability and safety of the products. As we show in the case of Saab EDS in section 4, these can be presented in two levels – one for the team and one for the management, or as we show in the case of Ericsson they can be part of a single indicator.

3 Elements of successful dashboards

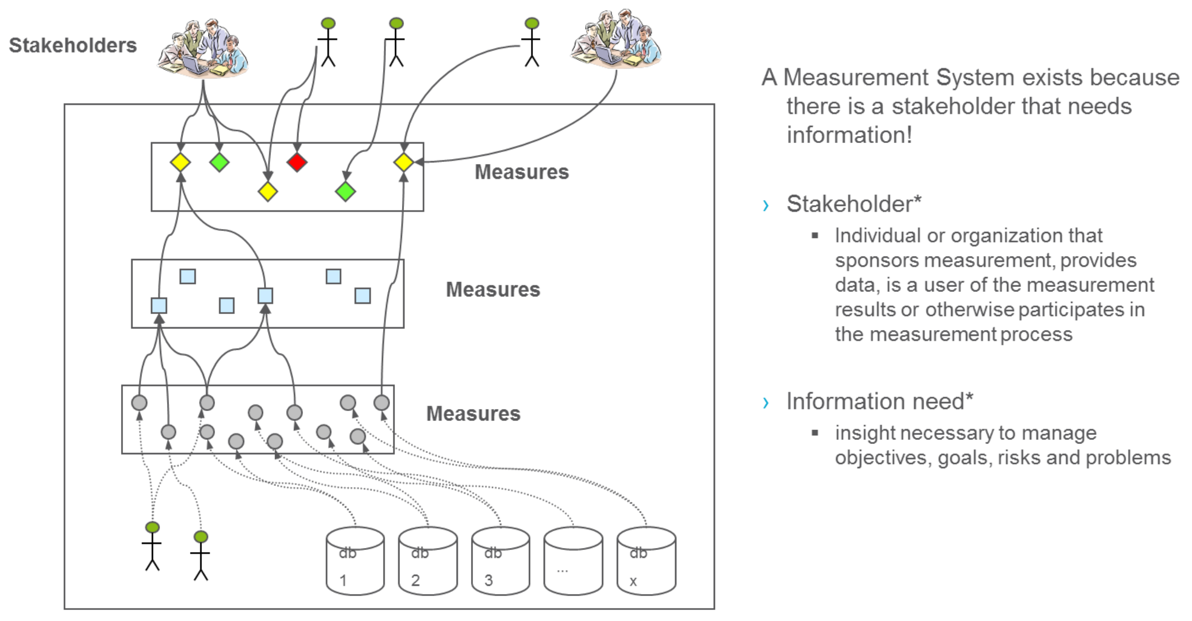
The indicators and measures used in the dashboards for monitoring quality of products under development often intend to support organizations in assessing the *release readiness* – i.e. when the product is to be ready to be released based on its functionality and quality. The indicators and measures are by far the most important element which determines whether the dashboard is successful. However, they have to be complemented with others to make the dashboard widely used at the company – standardization (section 3.1), early warning (section 3.2), focus on decisions and predictions (section 3.5), succinct presentation (section 3.3), and assuring information quality (section 3.4). The successful dashboards lead to actions by the team and the project towards achieving the set goals (e.g. delivery of features according to schedule, preventing unintended quality drops, monitoring the erosion of architecture).

3.1 Standardization

Using international standards increases the portability of measures and provides the possibility to show to customers that the measurement processes are planned and executed according to state-of-the-art practices. This is particularly important for the Agile and Lean organizations which usually have flexible planning which can be seen as risky for stakeholders used to plan-driven development. By using the standards we address the following challenge:

- *How to standardize the measures and indicators for monitoring development progress and release readiness taking into account product quality rather than schedule?*

This challenge needs to be address to show that the principles of agile planning are well-implemented in the company and that there is no risk for losing focus from the long-term product quality by focusing on flexible plans and immediate delivery of customer value. The most well-known standard for measurement processes in software and systems engineering is the European standard ISO/IEC 15939[1] (International Standard Organization and International Electrotechnical Commission, 2007) and the IEEE standard 1061-1998 (IEEE, 1998). The core component of the standard is the conceptual *measurement information model* which describes relationships between the main types of elements of measuring systems. Figure 2 presents an overview of this model from the standard and the two most relevant definitions for our chapter.



A Measurement System exists because there is a stakeholder that needs information!

› Stakeholder*
  - Individual or organization that sponsors measurement, provides data, is a user of the measurement results or otherwise participates in the measurement process

› Information need*
  - insight necessary to manage objectives, goals, risks and problems

Figure 2. Simplified view of the measurement information model from ISO/IEC 15939

[1] IEEE has adopted this standard under the same number – 15939-2008.

The information need is an insight necessary for a stakeholder to manage objectives, goals, risks, and problems observed in the measured objects. These measured objects can be entities like projects, organizations, software products, etc. characterized by a set of attributes. ISO/IEC 15939 includes the following definitions, which are relevant. The definitions were adopted from the more general standard – JCGM Vocabulary in metrology standard (International Bureau of Weights and Measures., 1993):

- Entity – object that is to be characterized by measuring its attributes.
- Attribute – property or characteristics of an entity that can be distinguished quantitatively or qualitatively by human or automated means.
- Base measure – measure defined in terms of an attribute and the method for quantifying it. This definition is based on the definition of base quantity from.
- Derived measure – measure that is defined as a function of two or more values of base measures. This definition is based on the definition of derived quantity from.
- Decision criteria – thresholds, targets, or patterns used to determine the need for action or further investigation, or to describe the level of confidence in a given result.
- **Indicator** – measure that provides an estimate or evaluation of specified attributes derived from a model with respect to defined information needs.
- Information product – one or more indicators and their associated interpretations that address an information need.

ISO/IEC 15939 is not the only framework available for structuring measures in measurement systems. The most well-known non-standard framework is Goal-Question-Metric (van Solingen and Berghout, 1999, Basili et al., 1994), which is used as a base for IEEE 1061-1998. The GQM framework is widely used in the research community and can replace ISO/IEC 15939 for structuring measures, but it does not recognize the concepts of stakeholders, information needs or indicators.

3.2 Focus on early warning

Knowing the status of the monitored entities at a particular point of time is naturally very important for the stakeholders. However, even more important is to know what the status will be a week, month or year ahead. This is particularly important for monitoring qualities of architectures since daily design decisions can have significant impact on subsequent space of design choices available for architects and designers. The stakeholders need indicators which warn about potential problems when the stakeholders still have the means to react and get their monitored entities back on track – for example to prevent architecture erosion. In other words, if early warning measures are "green" then the stakeholders can focus on monitoring the current progress, but if the early warning is "red" then the warning should be acted upon and it takes precedence. This requires specific kinds of measures and brings us to the following challenge:

- *What kind of measures and indicators should be used to monitor the development progress?*

From the experience we found the companies usually focus on quantifying the status as step 1 in introducing indicators and when this is in place the companies look for means of predicting the outcome based on the current situation. The most mature organizations focus also on indicators linked to simulations or "what-if" analyses (Scacchi, 1999). The analyses for architectures include performance simulations or architecture modifiability analyses – crucial for the Agile and Lean ways-of-working in the long run.

Examples of indicators which are used in early warning systems are:

- Release readiness – warning that the current development progress is not sufficient to meet the project plan
- Test execution progress – warning that the current test progress is not sufficient to assure that the quality goals are met
- Requirements breakdown progress – warning that there are not enough detailed requirements to continue software design during the current sprint for the team.
- Architecture rules violations, non-conformance, performance degradation

There are naturally more examples which we discuss based on the dashboards from each studied company.

3.3 Focus on triggering decisions and monitoring their implementation

The most important concepts from an external perspective on dashboards are *indicators* and *stakeholders*. The indicators are means for communicating the most important information about the current or predicted status of the monitored entities. The indicators are meant to trigger decision processes in the companies, including both formulating and executing decisions. This leads to the need to address the following challenge:

- *How to combine the need for indicators triggering decisions when needed with the need to collect larger number of measures for drill-down and monitoring of execution of decisions?*

In order to combine few indicators with many measures, a dashboard has to show indicators and link to a measurement system which collects measures and store them in a document/database (e.g. an associated Excel file). For example a successful indicator monitoring the complexity of the product attracts the attention of the stakeholder to the components which are too complex given pre-defined criteria. This triggers decisions from stakeholders for example to order additional resources to address the raising complexity in the product. By showing a trend in complexity development the stakeholders can observe whether the situations is getting out of control or when they should react.

The properly chosen stakeholders are crucial for the success in adopting the dashboard in a large organization. They must have the authority to define the indicator, its threshold (decision criteria and analysis model) and have the mandate to act upon the status of the indicator. For example the stakeholder for the release readiness indicator is the project manager who needs to constantly monitor when the product is ready to be released and has the mandate to take actions to meet the target release date if necessary – e.g. order overtime.

The stakeholders usually work closely with information providers who have the interest in communicating specific information, using measures and indicators. For example a software development team could be such a provider who assures that the team’s indicator objectively communicates the status of the development of a feature to the project management and to other interested entities like the line management. The team also needs to be notified about the status of the architecture of the product which can influence their planning.

3.4 Succinct visualization

In order to be effective in the communication of indicator's status to the stakeholders, the information products (according to ISO/IEC 15939) have to present the information in a compact manner. The modern tools usually support this by using apps for modern operating systems like Android, iOS, Windows or MacOS for presenting the status of the indicator and providing the entry point for more information like trends or statistics of base and derived measures (Staron et al., 2008). Using such simple means and focusing on presenting the most important information succinctly, addresses the following challenge:

- *How to visualize measures in order to effectively trigger decision processes?*

Dashboards usually contain 5-9 indicators and use metaphors of the same kind to convey the message of the status of the indicators. The metaphors are important for the presentation of the status of indicators (Johansson et al., 2007, Shollo et al., 2010) and based on the previous studies we could say that the most powerful metaphors are:

- Traffic lights – many successful indicators communicate three states: problem, warning, normal state. The traffic lights metaphor is perfect[2] for this kind of communication, especially if accompanied with the number to address the status of the indicator. For an example, please see Figure 4.
- Gauges and meters – when the focus of the indicator is primarily in the number rather than the state – using this metaphor is suitable as it provides gradation of the status, e.g. green close to yellow.

[2] The metaphor of a red light for problems showed itself to be very effective to attract attention. However, one should be very careful not to abuse the red color. If the decision criteria set the red color also when the status is not "red" then the stakeholders loose trust in this problem-warning signal.

The succinct presentation of the status of the indicator is combined with the ability (e.g. a hyperlink) to drill-down into detailed statistics to understand the cause of the status and act accordingly. The successful succinct indicator are usually supported by a number of base and derived measures directly used in calculating the status or needed to monitor that assumptions behind the analysis model and the indicator hold. The supporting base and derived measures are usually visualized as charts, e.g. trends.

3.5 Assuring information quality

In large software products the number of base and derived measures per indicator can be rather large (in some cases over 10 000 data points per indicator), which means that the probability that one data point is erroneous cannot be neglected. Since monitoring the architecture can comprise of multiple indicators (as is shown in section 4) and the information is spread to multiple teams, the quality of the calculations must be controlled automatically (to minimize the risk of making poor decisions by teams). In this section we elaborate on how the automated assessment of information quality addresses the following challenge:

- *How to assure high quality of information provided by measurement systems?*

Naturally, the more data is automatically processed, the more important the question about its quality becomes (Staron and Meding, 2009a). The stakeholders need to be informed whether the information in the dashboard is, for example:

- Up-to-date
- Calculated without errors
- Within pre-defined limits (e.g. the number of weeks to software release must be positive)

There are frameworks which characterize information quality in a quantitative way like the AIMQ framework (Lee et al., 2002). AIMQ provides a quality model for the information quality which includes the following examples of characteristics:

- Accessibility: the information is easily retrievable
- Completeness: the information includes all necessary values

- Concise representation: the information is formatted compactly
- Free of error: the information is correct
- Objectivity: the information was objectively collected
- Timeliness: the information is sufficiently current for our work

These quality attributes can be organized into two categories: (i) external quality – how the information is perceived by the stakeholder – semiotics of information, e.g. concise representation; (ii) internal quality – how the information is obtained and composed from components – *internals of dashboards*, e.g.: Free of error. Methods used for empirical validation of measures are used to assess the external information quality, e.g. case studies with indicators. The following work is particularly useful for this purpose: (Bellini et al., 2005, Raffo and Kellner, 2000, Stensrud et al., 2002, Yuming and Hareton, 2006, IEEE, 1998). The internal information quality can be checked during the run-time operation of measurement systems. Naturally a lot can be measured automatically, but there are limits to how much we can control. For example we cannot automatically check whether designers reported defects correctly in defect databases – since we do not "parse" the natural language in defect description we can only check that defects were reported and that the database was updated.

In section 4 we show how information quality is communicated to the stakeholders in practice in the case of Ericsson's release readiness dashboard.

4 Industrial dashboards

In this section we present how the three studied companies used the concept of release readiness in their dashboards for teams and for development projects. The approach of the three companies is different and based on the history of the use of measures in the companies and therefore not all elements of a successful dashboard are present in all companies. Table 1 presents a summary per company.

Table 1. Mapping of elements of successful dashboards per company

	Ericsson	**Volvo Car Corporation**	**Saab EDS**

Standardization	√	√	√
Focus on early warning	√	√	√
Focus on decisions and monitoring	√	√	√
Succinct presentation	√	√	√
Information quality	√		

Ericsson has been the company with the longest measurement experience w.r.t. standardized measurement processes in software engineering and therefore all elements are present in the dashboards at that company.

4.1 Companies

The companies operate in three different domains with its commonalities (e.g. embedded software development) and variability (e.g. development methodology, dependency on suppliers). The breadth of the companies and commonalities in the presented dashboard show that similar challenges and elements of successful dashboards are applicable for each domain.

4.1.1 Ericsson

Ericsson AB develops large software products for the mobile telecommunication network. The size of the organization during the study was several hundred engineers and the size of the projects was up to a few hundreds[3]. Projects were increasingly often executed according to the principles of Agile software development and Lean production system referred to as Streamline development (SD) within Ericsson (Tomaszewski et al., 2007). In this environment various teams were responsible for larger parts of the process compared to traditional processes: design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc.

[3] The exact size of the unit cannot be provided due to confidentiality reasons.

The organization used a number of measurement systems for controlling the software development project (per project) described above, a number of measurement systems to control the quality of products in field (per product) and a measurement system for monitoring the status of the organization at the top level. All measurement systems were developed using the in-house methods described in (Staron et al., 2010a, Staron et al., 2008), with the particular emphasis on models for design and deployment of measurement systems presented in (Staron and Meding, 2009b, Meding and Staron, 2009).

The needs of the organization had evolved from metric calculations and presentations (ca. 7 years before the writing of this paper) to using predictions, simulations, early warning systems and handling of vast quantities of data to steer organizations at different levels and providing information from project and line. These needs have been addressed by the action research projects conducted in the organization, since the 2006.

4.1.2 Volvo Car Corporation

Volvo Car Corporation (VCC) is a Swedish car original equipment manufacturer (OEM), based in Gothenburg. VCC was developing software and hardware in a distributed software development environment. For a number of Electronic Control Units (ECUs) software was developed in-house. The development was done by the software development teams who usually also had responsibility for integrating the software with the hardware developed by suppliers. The majority of the embedded software development in the car, however, was developed by external suppliers who design, implement and test the functionality based on specifications from VCC (Eklund et al., 2012, McGee et al., 2010).

The size of the entire automotive project in terms of resources was substantially larger than the projects in the telecom domain due to the fact that both OEMs and suppliers (first and second tier) were involved and car development projects were usually conducted using the product line approach with reference architectures (Gustavsson and Eklund, 2011). However we studied one team, which has a comparable size of a team at Ericsson and Saab EDS.

The studied organization at VCC was a software development team responsible for software for the ECU for Climate Control. The team was provided a set of measures to visualize the progress of development and communicate that upwards to the management.

4.1.3 Saab Electronic Defense Systems

Saab EDS developed embedded software and graphical user interfaces for ground based radar systems. The specific product we worked on was part of a larger product developed by several hundred developers, designers, testers, analysts etc. The historic project developing the product was driven in increments and did not utilize cross functional teams. The project management did some manual metrics on trouble reports.

The organization has since this project evolved into using more agile processes and cross functional teams. A lot of improvements and optimizations have also been done regarding software build and delivery times. Also to improve customer value, market competitiveness and profit, Saab AB Electronic Defense Systems in Gothenburg is going through a Lean transformation.

The organization at Saab Electronic Defense Systems has a history of using measures and communicating quality through dashboards. The dashboard presented in this chapter shows how the organization uses one measure – number of defects – in different granularity, to provide insight into the status of software development.

4.2 Dashboard at Ericsson

Ericsson chooses a dashboard which shows product release readiness in a compact form. The product release readiness indicator is intended to predict when the product under development has the appropriate quality for being released (Staron et al., 2012). The quality is measured in a number of open defect reports for the product – meaning that the right quality for releasing of the product is 0 defects. The defects can be related to the functionality of the software and its non-functional properties, in particular performance, reliability and availability. The properties are tested as part of regularly executed test suites.

The 0-defect criterion is sufficient only when another criterion is fulfilled – all functionality is tested and all test cases are passed. The stakeholder for this indicator is the project manager and the software development program is the group that has the need to communicate the information upwards to the management teams of the program and the organization. The indicator (RR, Release Readiness) has the following form:

$$RR = \left(\frac{\#defects}{defect\_removal\_rate - (test\_execution\_rate - test\_pass\_rate)}\right)$$

Where *#defects* is the number of open defects for the product[4], *defect_removal_rate* is the average number of removed defects during the last 4 weeks, *test_execution_rate* is the average number of test cases executed during the last 4 weeks and *test_pass_rate* is the average number of test cases passed during the last 4 weaks. The 4 weeks period is chosen based on statistical and empirical analyses. These analyses has shown that based on the length of the test cycles and the defect removal activities the 4 week period is the most appropriate length for this prediction and provides the most accurate results (Staron et al., 2012).

The formula is built from a number of base measures (e.g. test passed per week) and derived measures (e.g. test pass rate during the last 4 weeks), which shows how **standardization** according to ISO/IEC 15939 is realized in practice. Each of the measures in the formula is defined according to the standard with its measurement method and measurement function. The stakeholder has the means and ability to react and get the project back on track if needed.

Figure 3 presents how the indicator spreads in the organization on a daily basis – in a form of MS Vista Sidebar gadget (example of **succinct visualization**). It is the gadget which is the dashboard where the indicator is presented. It is complemented with an Excel file with trends for the measures in the formula for RR.

[4] This measurement included all defects that need to be removed from the product before the release.

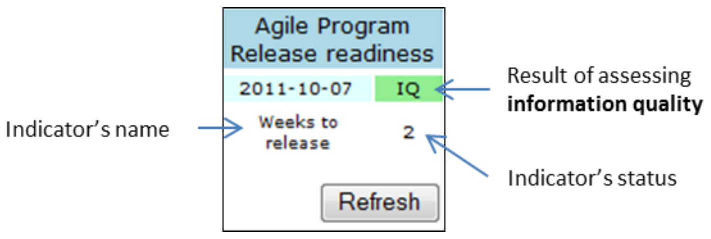

Agile Program Release readiness
2011-10-07
IQ
Weeks to release
2
Refresh
Result of assessing **information quality**
Indicator's name
Indicator's status

Figure 3. MS Vista Gadget with predicted week of release

The content of the gadget shows 2 weeks for the project to obtain the release-quality (*weeks to release)*. The presentation is simple and succinct giving the stakeholder (the manager of the studied product development project) the necessary information. The gadget also contains information quality assurance indicator (**assuring information quality**) which abbreviates to *IQ*, (Staron and Meding, 2009a). The details with base and derived measures are available in an associated MS Excel file once the gadget is clicked on.

In addition to the gadget the team has an auxiliary measurement system monitoring dependencies between architectural components – both explicit and implicit. The measurement system is based on monitoring of how software components change over time and which components change together (Staron et al., 2013) – with an example in Figure 4.

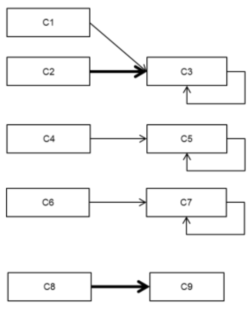


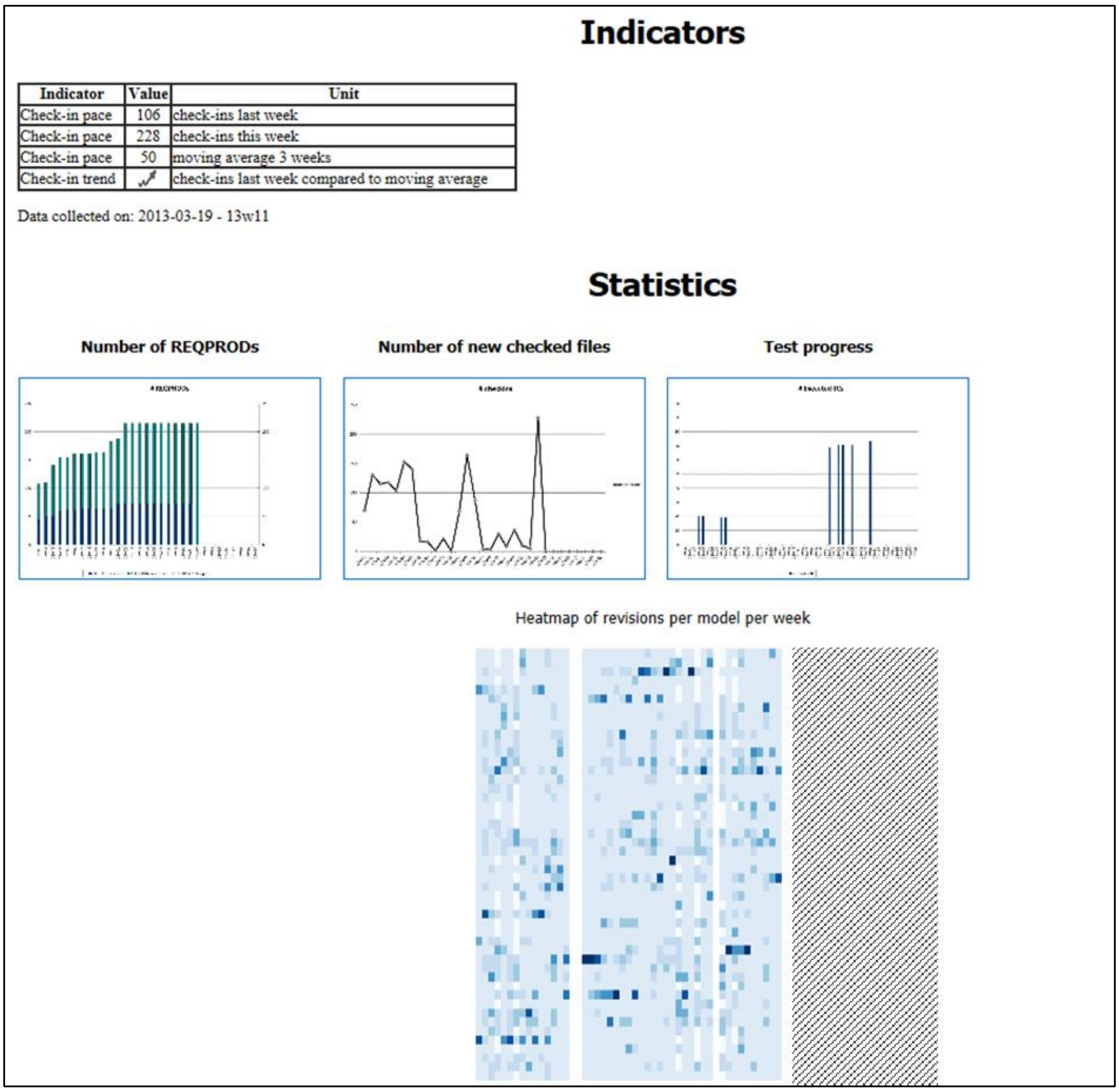
Figure 4. Implicit and explicit dependencies between architectural components (example of small set of components C1-C9).

The dependencies are visualized in a very succinct manner and allow teams to quickly find dependencies which are not explicit and can lead to architecture erosion. New dependencies cause updating the test strategies and thus influence the release readiness indicator (new test cases to execute, thus more time needed to test the product before the release.

4.3 Dashboard at Volvo Car Corporation

In the case of VCC we studied how one software development team can monitor development progress using a dashboard with three components – requirement management, version control for models and test progress. The team's interest was to communicate the status of the development of software for one ECU (Electronic Control Unit) for a family of modern cars within the Volvo brand. The dashboard was designed and developed together with a **stakeholder** from the team who had the insight into the development process and practices at the company. The stakeholder was designated by the team to represent the team's common view.

The dashboard for monitoring the development progress for the team is presented in Figure 5.



Indicators

Indicator	Value	Unit
Check-in pace	106	check-ins last week
Check-in pace	228	check-ins this week
Check-in pace	50	moving average 3 weeks
Check-in trend	↗	check-ins last week compared to moving average

Data collected on: 2013-03-19 - 13w11

Statistics

Number of REQPRODs

Number of new checked files

Test progress

Heatmap of revisions per model per week

Figure 5. Dashboard for monitoring quality of software product under development for one team

This dashboard presents indicators for monitoring the trend in software development by monitoring the pace of modeling of the ECU's functionality (the table under heading *Indicators*). Four indicators for check-in pace and trend have been chosen by the team to monitor that:

- **Check-in trend** – the main indicator capturing the readiness of functionality defined as the difference between the moving average and the number of check-ins last week – if the number of check-ins decreases w r t moving average then there is a high probability that the team is moving towards the testing phase. In a **succinct** way it captures the readiness of the team with the implementation of the functionality.
- **Number of check-ins last week** – the indicator captured the "temperature" of the development of the functionality.
- **Number of check-ins during the current week** – the indicator monitors the activities during the current week to support the team in monitoring how the status develops over time
- **Check-in pace** – the indicator showing what the average level (moving average) of check-ins is.
- **Heatmap of revisions per model per week** – visualizes how stable the architecture of the software is. This is a part of internal quality measurement for the team and helps to identify spots where architecture needs rework.

Since the team is working in Agile way with functional testing being an integral part of the development of models, the check-ins can be caused by new functionality being added or by changes of the existing functionality caused by defect fixes. The regression testing, however, was not included and therefore an additional measure was needed to control the progress of the basic quality assurance of the developed functionality.

The team monitored three auxiliary trends in the dashboard in order to control that the assumption of the indicators hold:

- Requirements elicitation and breakdown (Number of REQPRODs) – in order to reason about the release readiness the team assesses whether there are new requirements in the backlog. New requirements indicate that there is still new work to be done (in particular new updates to the models or new models to be developed).
- Modeling (Number of new checked files) – in order to reason whether the team is ready with the product they need to assess what the development trend is. The heatmap of model revisions shows the internal stability of the architecture.
- Test progress – the number of passed regression test cases.

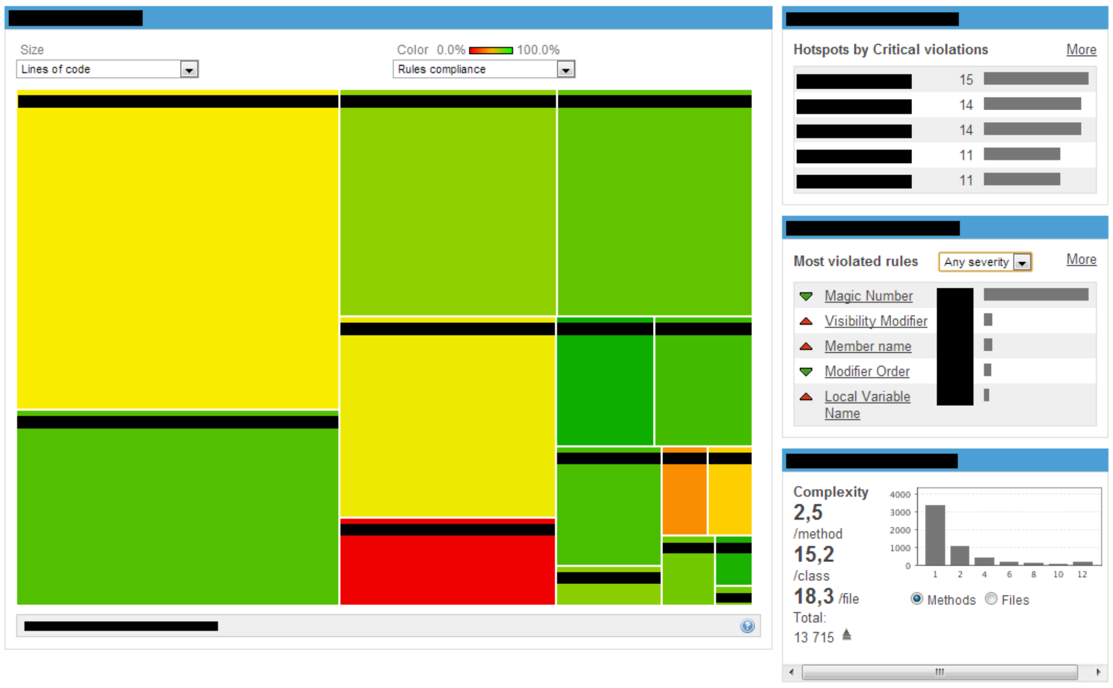
The dashboard presented in Figure 5 is built based on the same principles as the gadget presented in section 4.2, but the information is presented as a *flow* - three diagrams in one row in Figure 5 instead of a single indicator. This more exhaustive presentation of release readiness is motivated by the fact that the team wanted to have more insight in the development and communicate the status visually to stakeholders outside the team – e.g. sub-project managers.

The visual presentation of the information as a flow is also supposed to focus the team on **early warning**. By visualizing the trend in the number of new/changed requirements the team can monitor whether there is still functionality to be implemented and tested.

When the trend in the requirement growth is flat, the check-in trend is decreasing and the number of passed regression test cases is stable or growths, then the team is close to the release (**focus on decisions**).

4.4 Dashboard at Saab Electronic Defense Systems

At Saab EDS we studied two dashboards – one for monitoring the internal quality of software product under development (Figure 6) and one for monitoring the external quality (Figure 7). The dashboards are complemented with build radiators. The build radiators are used to show the current build status of products at the studied organization at Saab EDS. If a product is red (indicates a broken build) on the radiator people react to that and perform the actions needed to make it pass again.



Size: Lines of code

Color 0.0% 100.0%: Rules compliance

Hotspots by Critical violations

[redacted]	15
[redacted]	14
[redacted]	14
[redacted]	11
[redacted]	11

Most violated rules: Any severity

- Magic Number
- Visibility Modifier
- Member name
- Modifier Order
- Local Variable Name

Complexity 2,5 /method 15,2 /class 18,3 /file Total: 13 715

Figure 6. Dashboard for monitoring the internal quality (excerpt) at Saab EDS.

The dashboard in Figure 6 presents indicators for monitoring the current state and trend in internal software quality (ISO, 2005b) – in particular the complexity of the product. This dashboard is used on a daily basis among the developers as it visualizes data which is immediately important for them. The data for the indicators is generated by a tool for static code analysis and it shows what the status of such quality indicators is, e.g. what is the status of complexity of source code.

The early warning for problems is provided by other views in the same dashboard by showing trends for the metrics below (as an example):

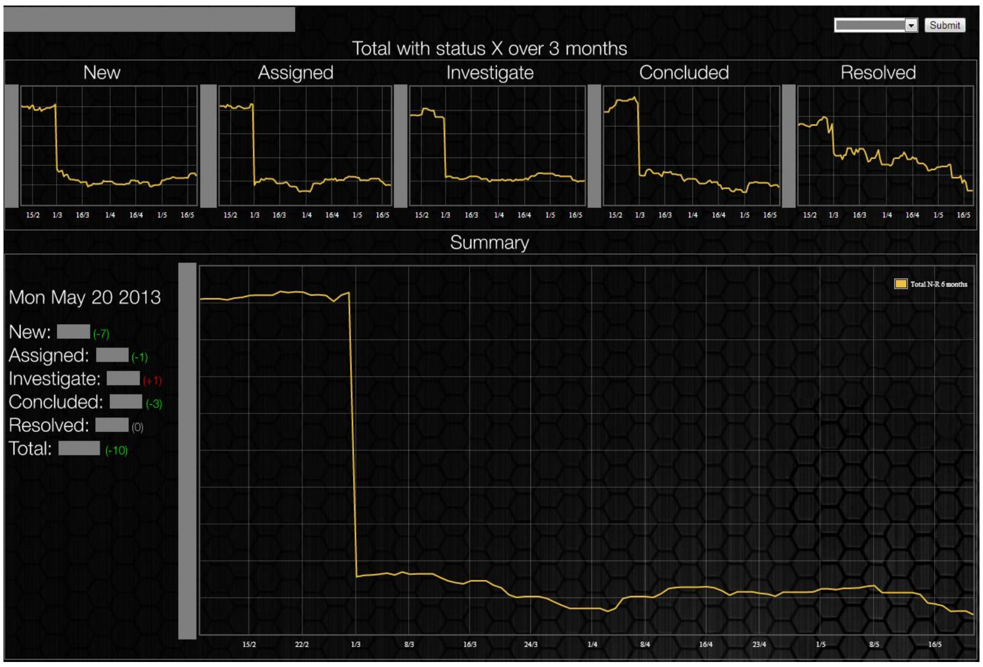
- The tree map/heat map (in the center of the figure with red, intensive color pointing attention towards the problem are and green color showing high quality of the area) view is used to identify software module with low rule compliance. The size of the rectangle is determined by lines of code. This information is used as input to decision making whether to clean up or rewrite a certain module[5].
- On top, right-hand side the dashboard shows hotspots by critical violations, which helps to quickly identify high risk code and quick wins. The hotspots are the most important parts of software

[5] Module names throughout the dashboard are grayed out for confidentiality reasons.

systems which violate the rules set by architects, e.g. non-conformance to architectural guidelines, interface visibility defects.

- The box in the middle to the right lists the most violated rules by severity. The grayed out field represent the number of occurrences and the distribution diagram to the left. This information is used to identify certain problem areas that should be addressed (focused code clean up, training etc).
- The bottom right box shows code complexity. Since a module with high complexity might be more prone to errors and also being harder to maintain, higher values demands actions to be taken for the module in question.

The dashboard presents a number of aspects of internal quality in a succinct way and is "clickable" in the sense that it allows the designers to drilldown into detailed measures or the same measures at lower abstraction levels (e.g. functions or parts of the code). The internal quality includes the majority attributes from ISO 25000 (the attributes from the parts of the standard which is approved by ISO). The attributes are calculated at multiple levels of the architecture – from the product level down to the module or function level (if applicable).



Mon May 20 2013
New: (-7)
Assigned: (-1)
Investigate: (+1)
Concluded: (-3)
Resolved: (0)
Total: (-10)

Figure 7. Dashboard for monitoring the external product quality at Saab EDS.

The dashboard[6] in Figure 7 presents indicators for monitoring trouble reports, i.e. the external quality of the software product under development. The information is used to easily identify bottlenecks in the flow of resolving the trouble reports. The small graphs represent the number of trouble reports in the corresponding state at a given time. The summary graph contains the sum of the small graphs at a given time. The colored numbers under summary indicates the trouble report delta over 24 hours.

An upgoing trend for resolved trouble reports could for example point towards lack of testing resources.

5 Recommendations for other companies

Based on the experience with constructing dashboards for various companies, we can identify a number of recommendations for companies which intend to introduce dashboards. We divide the recommendations into two parts: constructing the dashboards (process) and indicators and measures (product).

5.1 Recommendations for constructing the dashboards

To be successful in introducing a dashboard we recommend the companies to:

- **When defining indicators, choose the indicators that match stakeholders who have the mandate and means to react upon the status of the indicators**. It is imperative that the stakeholders have the mandate to act. Without this the dashboards is just a nice chart which perhaps can be used for discussions. With the stakeholder who has the mandate and means the indicators become a very powerful tool for the organization to steer itself based on facts. For monitoring quality of product under development the stakeholders should be product managers and project managers. They have the mandate to order extra resources to address quality issues and to rework the architecture; they can also give assignments to architects and designers to address specific problems with architecture.
- **Limit the number of indicators**: Depending on the company's awareness and trust in measures the dashboards should contain the appropriate number of indicators. If there is a well-established

[6] Actual numbers and organization name has been grayed out.

measurement culture in the organization, the measurement teams have a strong reputation in providing reliable measurement products, then it is enough with one indicator to communicate efficiently. However, if the organization is still gaining the awareness and the measurement team is gaining reputation, then there should be more indicators (although no more than 10 for the sake of coherence). The number of indicators usually decreases over time as organizations become more aware of the inherent dependencies between the initial set of indicators. By assigning the right test cases companies can monitor qualities of the product (e.g. performance or reliability) by monitoring the test progress, which is one example of combining two inter-dependent indicators.

- **Involve multiple parts of the organization, at least product management, product development and verification and validation (V&V)**: Even when defining indicators for smaller entities the company should keep in mind that no entity is isolated. When the companies work with products they need to understand what are the premises for the product (features, markets – captured by product management), software development and requirements of the company's quality assurance (testing at different levels, customer feedback – captured by V&V). Having all three – product management, development and V&V – allows the company to focus on the end result rather than sub-optimizing towards smaller processes. This observation is particularly important for monitoring the quality attributes of the architecture since problems with architecture can influence the extensibility of the product (thus also agility of the company), the performance of the product or the quality assurance effort (thus cost of the quality).

5.2 Recommendations for choosing indicators and measures

To choose the right indicators and measures for the dashboards for monitoring the quality of products under development we recommend the companies to:

- **Focus on product**: The focus of the successful indicator is usually on the monitored entity, which can be a product, a process or a project. However, in reality companies focus on products and the profitability of the companies, which demands that the indicators also focus on products. Problems with processes or projects will be visible on the product level (e.g. architecture erosion, low quality, delays in delivery) and they have to be monitored using base/derived measures as trends. However, since companies do not sell their processes or projects, the indicators should be related to products. The RR indicator at Ericsson shows how a set of process and product measures (e.g. test progress) can be packaged into a product-focused indicator. Various test cases are supposed to test both functional and non-functional properties, but no gradation is done on which is more important.

- **Focus on end result**: The main focus of indicators for monitoring the quality of products under development should be on the end product, i.e. software or software-hardware system and not on individual artifacts. Since it is the end product that the software organizations provide to their customers, it is the end product which should have the right quality and functionality. Therefore, monitoring particular artifacts like requirements or architecture is not sufficient and lead to sub-optimizations and short-sighted decisions.

6 Further reading

The dashboards for monitoring the development progress are related to monitoring bottlenecks in large software development organizations which has been studied by Staron and Meding (Staron and Meding, 2011) and Pedersen and Wohlin (Petersen and Wohlin, 2011). Monitoring bottlenecks is a useful method for monitoring the capacity of the organization. In the previous work we developed and introduced a method based on automated indicators for monitoring the capacity and bottlenecks in the work flow in the large software development project – i.e. the process/project view of software development in that organization.

The very rudimentary and effective first measure to be used by Agile teams to monitor quality is RTF (Running Tested Features) measure, popular in XP (Jeffries, 2004). The metric combines three important concepts – the feature (i.e. a piece of code useful for the end-user, not a small increment that is not visible to the end user), execution (i.e. adding the value to the product through shipping the features to the customer), and the testing process (i.e. the quality of the feature – not only should it be execute, but also be of sufficient quality). This measure stimulates smart continuous deployment strategies and is intended to capture similar aspects as our release readiness indicator although in smaller projects.

Monitoring the trends in RTF can provide indications of architecture erosion over time (RTF becoming longer over time).

A set of other metrics useful in the context of continuous deployment can be found in the work of Fritz (Fitz, 2009) in the context of market driven software development organization. The metrics presented by Fritz measure such aspects as continuous integration pace or the pace of delivery of features to the customers. These measures complement the indicators presented in this paper with a different perspective important for product management.

The delivery strategy which is an extension of the concept of continuous deployment has been found as one of the three key aspects important for Agile software development organizations in a survey of 109 companies by Chow and Cao (Chow and Cao, 2008). The indicator presented in this paper is a means of supporting organizations in their transition towards achieving efficient delivery processes which are in line with the delivery strategy prioritized by practitioners in this survey.

The view on indicators, stakeholders and measures presented in ISO/IEC 15939 is consistent with other engineering disciplines, the standard states that it is based on ISO/IEC 15288:2007 (Systems and software engineering - System life cycle processes), ISO/IEC 14598-1:1999 (Information technology - Software product evaluation) (International Standard Organization, 1999), ISO/IEC 9126-x (International Standard Organization and International Electrotechnical Commission, 2001), ISO/IEC 25000 series of standards, or International vocabulary of basic and general terms in metrology (VIM) (International Bureau of Weights and Measures., 1993). These standards are recommended for companies aiming at successful use of dashboards for monitoring quality of products under development.

Finally, readers interested in more details about constructing dashboards based on ISO 15939 can find more information about automation, frameworks and modeling of measures can find more information in (Staron et al., 2008, Staron et al., 2010a).

7 Conclusions

Qualities and architecture of modern software products are distinguishing excellent products from the average ones and can influence the success of software development companies. Extensible and scalable architectures of products combined with Agile and Lean software development can determine a long-term sustainable businesses and maximize the value which customers get from these products. Agile and Lean software development methodologies change the way in which organizations work with monitoring of quality of software products under development and assessment of release readiness. Continuous development of software using these methodologies demands continuous quality management. The inherent dependability of these methodologies on self-organized teams shifts the focus from monitoring quality from the higher management to communicating the quality status. The communicated quality status triggers decision processes on all levels in the organizations and new ways of monitoring the progress of the implementation of these decisions emerge.

In this chapter we investigated how three organizations monitor the quality of software products during development. We explored how Ericsson in Sweden works with one effective indicator of *release readiness* to monitor the status and trigger decisions. We studied how Saab Electronic Defense Systems combines trends for defect management with monitoring internal quality of products. Finally we also studied how a team at VCC in Sweden can communicate the software development progress as an alternative to manual status reporting. We provided recommendations for other companies willing to use dashboards – how to construct them and how to choose indicators and measures.

References

BASILI, V., CALDIERA, G. & ROMBACH, H. D. 1994. The Goal Question Metric Approach. Available: ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf.

BELLINI, P., BRUNO, I., NESI, P. & ROGAI, D. Comparing fault-proneness estimation models. 2005. 205-214.

BUSE, R. P. L. & ZIMMERMANN, T. 2012. Information Needs for Software Development Analytics. *34th International Conference on Software Engineering (ICSE 2012 SEIP Track).* Zurich, Switzerland: Microsoft Research Report.

CHOW, T. & CAO, D.-B. 2008. A survey study of critical success factors in agile software projects. *Journal of Systems and Software,* 81**,** 961-971.

EKLUND, U., JONSSON, N., ERIKSSON, A. & BOSCH, J. A reference architecture template for software-intensive embedded systems. Proceedings of the WICSA/ECSA 2012 Companion Volume, 2012. ACM, 104-111.

FITZ, T. 2009. *Continuous Deployment at IMVU: Doing the impossible fifty times a day* [Online]. Available: http://timothyfitz.wordpress.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/.

GUSTAVSSON, H. & EKLUND, U. 2011. Architecting automotive product lines: industrial practice. *Software Product Lines: Going Beyond*, 92-105.

IEEE 1998. IEEE standard for a software quality metrics methodology. *IEEE Std 1061-1998*.

INTERNATIONAL BUREAU OF WEIGHTS AND MEASURES. 1993. *International vocabulary of basic and general terms in metrology = Vocabulaire international des termes fondamentaux et généraux de métrologie,* Genève, Switzerland, International Organization for Standardization.

INTERNATIONAL STANDARD ORGANIZATION 1999. *Information technology -- Software product evaluation 14598-1:1999*.

INTERNATIONAL STANDARD ORGANIZATION & INTERNATIONAL ELECTROTECHNICAL COMMISSION 2001. ISO/IEC 9126 - Software engineering – Product quality Part: 1 Quality model. Geneva: International Standard Organization / International Electrotechnical Commission.

INTERNATIONAL STANDARD ORGANIZATION & INTERNATIONAL ELECTROTECHNICAL COMMISSION 2007. ISO/IEC 15939 Software engineering – Software measurement process. Geneva: International Standard Organization / International Electrotechnical Commission,.

ISO, B. 2005a. 9000: 2005 Quality management systems. Fundamentals and vocabulary. *British Standards Institution*.

ISO, I. 2005b. IEC 25000 Software and system engineering–Software product Quality Requirements and Evaluation (SQuaRE)–Guide to SQuaRE. *International Organization for Standarization*.

JEFFRIES, R. 2004. *A Metric Leading to Agility* [Online]. xprogramming.com. Available: http://xprogramming.com/xpmag/jatRtsMetric 2011].

JOHANSSON, L., STARON, M. & MEDING, W. 2007. An Industrial Case Study on Visualization of Dependencies between Software Measurements. *In:* ARTS, T. (ed.) *Software Engineering and Practice in Sweden.* Göteborg, Sweden.

LEE, Y. W., STRONG, D. M., KAHN, B. K. & WANG, R. Y. 2002. AIMQ: a methodology for information quality assessment. *Information & Management,* 40**,** 133-146.

MCGEE, R. A., EKLUND, U. & LUNDIN, M. Stakeholder identification and quality attribute prioritization for a global Vehicle Control System. Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, 2010. ACM, 43-48.

MEDING, W. & STARON, M. 2009. The Role of Design and Implementation Models in Establishing Mature Measurement Programs. *In:* PELTONEN, J. (ed.) *Nordic Workshop on Model Driven Engineering.* Tampere, Finland: Tampere University of Technology.

PETERSEN, K. & WOHLIN, C. 2011. Measuring the flow in lean software development. *Software: Practice and Experience,* 41**,** 975-996.

PHAAL, R., FARRUKH, C. J. P. & PROBERT, D. R. 2004. Technology roadmapping - A planning framework for evolution and revolution. *Technological Forecasting and Social Change,* 71**,** 5-26.

POPPENDIECK, M. & POPPENDIECK, T. 2007. *Implementing Lean Software Development: From Concept to Cash,* Boston, MA, USA, Addison-Wesley.

RAFFO, D. M. & KELLNER, M. I. 2000. Empirical analysis in software process simulation modeling. *Journal of Systems and Software,* 53**,** 31-41.

RUHE, G. 2003. Software Engineering Decision Support – A New Paradigm for Learning Software Organizations. *In:* HENNINGER, S. & MAURER, F. (eds.) *Advances in Learning Software Organizations.* Springer Berlin / Heidelberg.

RUHE, G. & SALIU, M. O. 2005. The art and science of software release planning. *Software, IEEE,* 22**,** 47-53.

SANGAL, N., JORDAN, E., SINHA, V. & JACKSON, D. 2005. Using dependency models to manage complex software architecture. *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.* San Diego, CA, USA: ACM.

SCACCHI, W. 1999. Experience with software process simulation and modeling. *Journal of Systems and Software,* 46**,** 183-192.

SHARP, H., BADDOO, N., BEECHAM, S., HALL, T. & ROBINSON, H. 2009. Models of motivation in software engineering. *Information and Software Technology,* 51**,** 219-233.

SHOLLO, A., PANDAZO, K., STARON, M. & MEDING, W. Presenting Software Metrics Indicators: A Case Study. The 20th International Workshop on Software Measurement IWSM 2010, 2010.

STARON, M. & MEDING, W. Ensuring Reliability of Information Provided by Measurement Systems. *In:* ABRAN, A., BRAUNGARTEN, R., DUMKE, R., CUADRADO-GALLEGO, J. & BRUNEKREEF, J., eds. Software Process and Product Measurement, 2009a. Springer Berlin / Heidelberg, 1-16.

STARON, M. & MEDING, W. 2009b. Using Models to Develop Measurement Systems: A Method and Its Industrial Use. *In:* ABRAN, A., BRAUNGARTEN, R., DUMKE, R., CUADRADO-GALLEGO, J. & BRUNEKREEF, J. (eds.) *Software Process and Product Measurement.* Amsterdam, NL: Springer Berlin / Heidelberg.

STARON, M. & MEDING, W. Monitoring Bottlenecks in Agile and Lean Software Development Projects – A Method and Its Industrial Use. *In:* CAIVANO, D., OIVO, M., BALDASSARRE, M. & VISAGGIO, G., eds. Product-Focused Software Process Improvement, 2011 Tore Cane, Italy. Springer Berlin / Heidelberg, 3-16.

STARON, M., MEDING, W., HANSSON, J., HÖGLUND, C., ERIKSSON, P. & NILSSON, J. 2013. Identifying Implicit Architectural Dependencies using Measures of Source Code Change Waves. *Software Engineering and Advanced Aplications (39th International Conference).* Santander, Spain.

STARON, M., MEDING, W., KARLSSON, G. & NILSSON, C. 2010a. Developing measurement systems: an industrial case study. *Journal of Software Maintenance and Evolution: Research and Practice*, n/a-n/a.

STARON, M., MEDING, W. & NILSSON, C. 2008. A Framework for Developing Measurement Systems and Its Industrial Evaluation. *Information and Software Technology,* 51**,** 721-737.

STARON, M., MEDING, W. & PALM, K. 2012. Release Readiness Indicator for Mature Agile and Lean Software Development Projects. *Agile Processes in Software Engineering and Extreme Programming***,** 93-107.

STARON, M., MEDING, W. & SÖDERQVIST, B. 2010b. A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. *Information and Software Technology,* 52**,** 1069-1079.

STENSRUD, E., FOSS, T., KITCHENHAM, B. & MYRTVEIT, I. An empirical validation of the relationship between the magnitude of relative error and project size. 2002. 3-12.

TOMASZEWSKI, P., BERANDER, P. & DAMM, L.-O. 2007. From Traditional to Streamline Development - Opportunities and Challenges. *Software Process Improvement and Practice,* 2007**,** 1-20.

VAN SOLINGEN, R. & BERGHOUT, E. 1999. *The Goal/Question/Metric Method. A Practical Guide for Quality Improvement of Software Development,* London, McGraw-Hill.

WARD-DUTTON, N. 2011. Software Econometrics: Challenging assumptions about software delivery. *IBM.com podcast companion report* [Online].

YUMING, Z. & HARETON, L. 2006. Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *Software Engineering, IEEE Transactions on,* 32**,** 771-789.