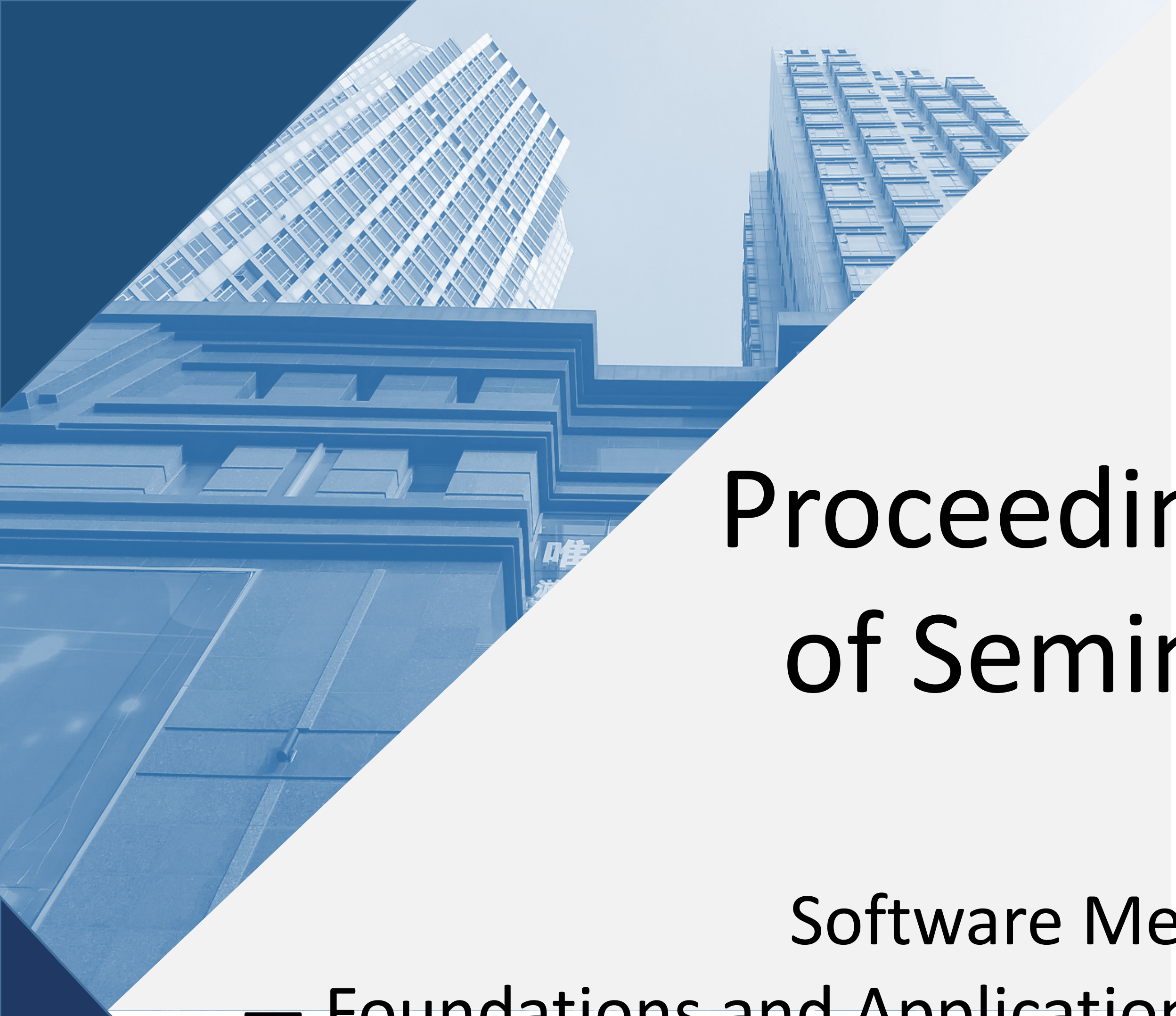


Proceedings of Seminar

Software Metrics — Foundations and Applications —

2010

Editors: Horst Lichter
Matthias Vianden

SWC Software Construction

RWTH AACHEN UNIVERSITY

Contents

Part I

Metric Process

Chapter 1

IT Controlling

Mareike Bültmann

Contents

Abstract: IT-Controlling has to deal with the challenge of intransparency and old-fashioned controlling methods applied to the IT-sector. The target of current IT-Controlling is to integrate IT into the whole business controlling process as a part of shifting IT-Controlling from the operational aspects to an established strategical oriented IT-Controlling. Therefore the Balanced Scorecard as a strategical management system can help to measure performance as well as to monitor goal achievement. This work describes different approaches of applying the BSC for the purpose of IT-Controlling and explains certain metrics which are relevant for the specific BSC of IT.

1.1 Introduction

IT-Controlling in the past was strongly cost related and sometimes misunderstood as cost-reduction. IT is integrated everywhere in the business processes, and costs are the easiest thing to measure and compare. But IT is much more complex and needs to be controlled in a right manner. IT-Controlling has to deal with insufficient transparency and has to provide methods to overcome the intransparencies. It has to be linked with performance-orientation and enhancing efficiency [Gad05].

Regarding IT-Controlling, different questions arise in companies, like ”Is that investment in IT really worthwhile?”, ”Is that implemented IT-application successful?”, ”Is the IT department productive and/or effective?”, and ”Should IT be outsourced?”. Hence, it is imperative to measure the value and to evaluate the performance in order to achieve business goals with IT [MDT99]. Two additional questions dealing with measurement however are also important: ”doing the right thing” and ”doing it right” (see section 1.2.2). These are the main subjects when dealing with performance and goal achievement measures, which are the focus of the Balanced Scorecard. The BSC supports the process of getting from an operational point of view to a strategical one.

This paper provides an overview of IT-Controlling, where the aims and tasks as well as the difference between strategical and operational IT-Controlling are explained. Also, different subcategories of IT-Controlling are described. Afterwards, the Balanced Scorecard is introduced as an instrument for measurement in the IT-Controlling. Different approaches of applying a BSC for IT are depicted, and some possible (used) metrics are mentioned.

1.2 Background of IT-Controlling

IT-Controlling can be seen as a subsystem of the whole controlling system. The aim of the management is to develop a mission and to realize the corporate goals of long-term benefit. For supporting these managerial functions, controlling has to provide a planning and controlling system for determining the aims and strategies as well as checking the achievements of the objectives. Therefore, data is provided, prepared and presented. This approach can easily be transformed into IT-Controlling. It only has to be adapted to the specific needs of IT-Controlling [KÖ5].

Looking at the concept of IT-Controlling, there are different interpretations, depending on the point of view. They have in common that IT-Controlling must be integrated into the controlling of the entire business of the company. Therefore, it is important to look at the IT-department not only as a part of the cost driver but also as a part of the operation efficiency of the company.

Out of the many given possible definitions, a well-established one is that IT-Controlling is a functional and department overlapping coordination system for the IT division and the information science of the whole business [KÖ5].

1.2.1 Tasks and Aims of IT-Controlling

The tasks and aims of controlling in the whole company can be applied to IT-Controlling. This also covers transparency of several domains: Transparency of the business strategy, of the finances and of the processes for reaching better economic efficiency. As a subsystem of the management system, it must ensure the decision making of the management. Therefore it moderates the management process of finding objectives, planning and supervision. It supplies the decision-maker with data and information to assure the leadership ability and also organizes the future-oriented reporting [KÖ5].

Also, IT-Controlling has to monitor the effectiveness and efficiency of the IT in the company to maximize its benefits. It must be realized in a flexible manner to have the possibility to react fast enough on changes. IT-Controlling focuses on providing IT services and coordinating the usage of resources in this department, therefore the processes and projects need to be monitored and governed [KÖ5].

1.2.2 Strategical and operational IT-Controlling

IT-Controlling is divided into strategical and operational IT-Controlling, where the strategical IT-Controlling deals with the increase of effectiveness, while the operational IT-Controlling deals with increasing the efficiency. In detail, the **strategical IT-Controlling** focuses on the objectives of the whole company, without any time constraints. The core question of strategical IT-Controlling is: **"to do the right things"**. Thus IT has to support the achievement of the business goals as a competitive factor in a strategical frame of the company. The **operational IT-Controlling** has to deal efficiently with the measures of the strategical controlling. The core question of the operational IT-Controlling is: **"to do the things right"**. It only works within a given time frame and looks at selected processes, information systems or cost units. The application domains of the metrics and tools are: Profit, liquidity and productivity, either of the entire company or of IT projects [Gad05].

The IT-Controlling has to provide the optimal contribution to the added value of the company concerning the IT. Thus, the corporate strategy and objectives must guarantee that the IT-Governance (see section 1.2.3) provides a value-oriented focus on IT. The IT has to be understood as a supporting process of the whole business process, where the strategical alignment of the IT-Controlling is important. To achieve long-term objectives, the IT-Governance must be process related and value oriented. Therefore the strategy and objectives of the company are in the focus of IT-Controlling [Gad05].

1.2.3 Variants/kinds of IT-Controlling

Concerning the different tasks of IT-Controlling, three different subcategories of IT-Controlling can be derived (see figure 1.1). These subcategories are the

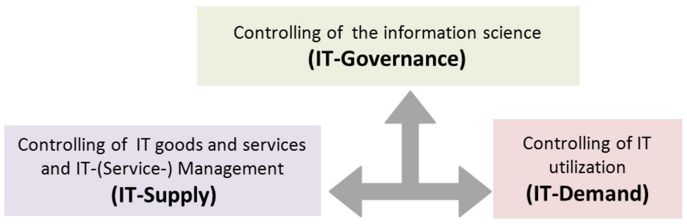

Controlling of the information science
(IT-Governance)
Controlling of IT goods and services and IT-(Service-) Management
(IT-Supply)
Controlling of IT utilization
(IT-Demand)

Figure 1.1: The different subcategories of IT-Controlling [K05]

result of a different kind of specification. First of all, IT-Controlling is needed in those departments that perform IT-services **(IT-Supply)**. The focus is on the "'production"' of IT-services. Also, departments that are using the IT-services need to be controlled, because there are different IT-resources which are used in the different departments **(IT-Demand)**. The entire organization needs a superordinated IT-Controlling for the whole information science. To get the IT-Supply and IT-Demand in balance, the **IT-Governance** is needed [K05].

IT-Supply

IT-Supply deals with the supply of services and goods. The maintained services have to be coordinated to react on the demand of the other business units or customers. The IT-business units have to provide a supply portfolio where they have to know which services, kind of services and sets of services will be demanded. The IT-systems with the applications and the infrastructure are also part of the IT-Supply, because the functionality, usability and operational reliability of the systems are also a part of the IT services. The task of IT-Controlling, concerning the services, is to assist the management in analyzing the market potential, deriving the service supply, designing the service portfolio and supporting the planning of the quantities, the costs and price calculation. Regarding processes and projects, it is important to know the costs and needed resources. IT-Controlling tasks concerning these aspects are to coordinate the different processes of supply. Furthermore, additional tools have to be supplied to delegate the processes. The performance, the costs per unit and the process costs have to be monitored. Concerning the IT-system, the IT-Controlling has to support the management in analyzing and evaluating the system portfolio (which contains the application and infrastructure). Furthermore, it should help in analyzing the complexity, the functionality, the redundancy in the system and the investment needs. Also IT-Controlling has to consider the effectiveness of the supply, the profits and benefits settings, the Make or Buy Decision, the fixed and indirect costs and the coordination [Kö05].

IT-Demand

IT-Demand deals with the usage of IT-services and IT-resources. Target of controlling the IT-Demand is the economic effectiveness of the usage of IT-services: The demand of IT has to be investigated, where you can distinguish between efficiency improvement, business reengineering and business enabling. Efficiency can be improved by accelerating the work flow and therefore reducing the costs. By restructuring the processes and the organization while the output stays the same, the appliance of IT can radically change, simplify and accelerate processes and therefore also have positive effects on costs. To measure or to estimate the generated benefit is task of the IT-Controlling. By business enabling, new services and products are created. The tasks here are to evaluate the advantages of IT and to detect the caused costs [Kö5].

IT-Governance

IT-Governance is the instance that controls the different aspects of IT-Controlling to achieve the aims of the management: Regarding the IT-Demand and IT-Supply, the different portfolios of service supply and demand have to be coordinated to optimize the whole process and therefore coordinate and merge these portfolios to gain an optimal overall efficiency. The controlling of the information science (data and information in the company) is superordinated to the IT-Supply and IT-Demand. The goal is to optimize the use of IT from the point of view of the whole company. The generated benefits must be bigger than the caused costs. [KÖ5].

1.2.4 Problems

It is hard to reconcile these different aspects of IT-Controlling. The different subcategories have different kinds of performance drivers. So an instrument is needed which manages the aspects in order to improve and support IT-Controlling. One core idea should be to get a more transparent and flexible IT-Controlling: The given business strategies should be monitored and controlled to gain effectiveness and efficiency. To solve the problems of IT-Controlling and gain an adequate performance measurement, the IT department has to follow the strategical targets to realize the recommended IT-strategy. It has to know which measure belongs into which subcategory of IT-Controlling or the IT department and has to document the effects by means of measures and performance indicators. To integrate these performance measures into the whole process, the process needs to be standardized and centralized. IT is part of the business and therefore needs to be monitored within this system [RK07]. Specific concepts of ratio systems already exist; these systems often start with a key performance indicator and then try to deduct a system of sub performance indicators. But their construction is static and cannot be adapted to changes, furthermore it looks only at the indicators and not at the relations [MGB02].

Also the traditional IT-Controlling is technical and cost oriented. It measures the technical performance and costs to help the accounting of services for IT-systems. But with the increasing intensity of assignment of IT, more and more tasks must be realized by the IT-Controlling. It has to include aspects like qualitative values leading to an overall reporting system with integral operating figures. The aim is to realize a controlling which is not strictly operational oriented. Until now, the IT-reporting is separated from the current strategical needs of the company. The operational values are generated but cannot be interpreted without any strategical reference. An uniform IT-reporting is missing: The IT data reported from different departments are not merged. [RK07].

1.3 A Tool for IT-Controlling: The Balanced Scorecard (BSC)

The BSC is a management tool to measure the strategic performance and goal achievements. It was first introduced by Robert S. Kaplan and David P. Norton in the 1990s. It can be used to keep track of the execution of activities by staff and monitor the consequences that come along with these actions. The core idea of the BSC is the mixture of financial and non-financial measures whose values are the operationalization of the strategy of the company. It consists of reports for each perspective where the main metrics are used to represent the relevant information. The ”classical” perspectives are: Financial, Customer, Internal Business Process and Learning and Growth [KN96].

The BSC systematically helps to get a strategical oriented controlling with a smaller focus on the short term surveillance of operational values [MGB02] [RK07]. It helps to come along with the prior cost orientation and with the lack of an integrated IT-reporting (see section 1.2.4). Furthermore, the BSC helps the IT-Controlling to gain an adequate performance measurement. In detail, the BSC has an open structure: it can flexibly be adapted to new circumstances and changes, which was not possible with static ratio systems [MGB02].

1.3.1 The Structure of the Balanced Scorecard

The BSC is accompanied by a complex design process and reports for four perspectives where measures have to be defined and linked.

The design process

It relies on four processes to bind short-term activities to long-term objectives. First of all, an organization vision must be translated into values and measures. Second, the strategy must be available for everyone, therefore it must be communicated and linked. The strategy must also be integrated into the business and financial plans. Afterwards the results must be examined and eventually the objectives and measures from the BSC must be modified. The design process is shown in the following figure.

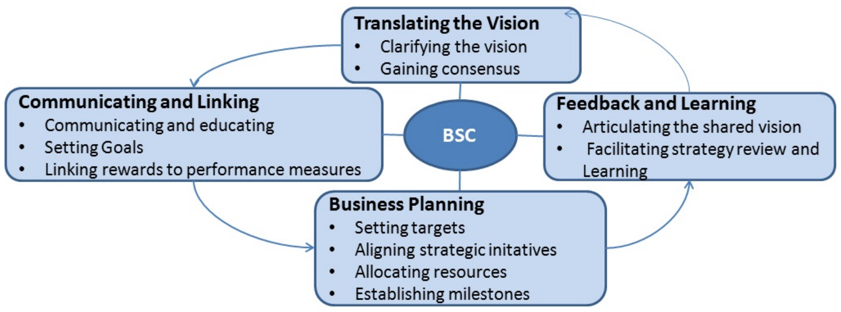

Translating the Vision
• Clarifying the vision
• Gaining consensus
Communicating and Linking
• Communicating and educating
• Setting Goals
• Linking rewards to performance measures
BSC
Feedback and Learning
• Articulating the shared vision
• Facilitating strategy review and Learning
Business Planning
• Setting targets
• Aligning strategic initatives
• Allocating resources
• Establishing milestones

Figure 1.2: The design process of the BSC [KN96]

The first process **(Translating the vision)** helps managers to build their own consensus vision and strategy. It forces the management to come to an agreement on the metrics used to operationalize the visions, because the statements in vision and strategy must be expressed as an integrated set of objectives and measures that describe the long-term drives of success. In the second process **(Communicating and Linking)**, the managers communicate their strategy up and down the organization and link it to departmental and individual objectives. By using the BSC, all levels of the organization understand the long-term strategy. In the next process, **(Business Planning)**, the business and financial plans are integrated. By using BSC, the organization can coordinate the allocation of resources and the setting of priorities for time, energy and resources and therefore move towards their long term strategic objectives. In the fourth process, **(Feedback and learning)**, strategic learning is applied. The short-term results from the four perspectives (see section 1.3.1) are used to optimize the strategies by continously iterating the four processes [KN96].

Description of the perspectives

The classical BSC consists of four fundamental perspectives, as shown in the figure below: In the **financial perspective** (shareholder view), the mission is to

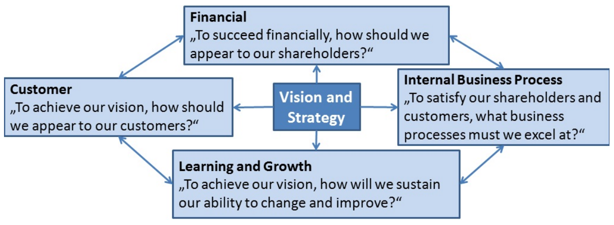

Financial
„To succeed financially, how should we appear to our shareholders?“
Customer
„To achieve our vision, how should we appear to our customers?“
Vision and Strategy
Internal Business Process
„To satisfy our shareholders and customers, what business processes must we excel at?“
Learning and Growth
„To achieve our vision, how will we sustain our ability to change and improve?“

Figure 1.3: The perspectives of the BSC [KN96]

succeed financially by delivering value to the shareholders. Therefore the utility and the costs are measured. In the **customer perspective** (value-adding view), the mission is to achieve the vision by delivering value to the customers. In the **internal business perspective** (process based view), the mission is to satisfy the shareholders and customers by promoting efficiency and effectiveness of the business processes. Therefore the goods and services have to be organized optimally. In the **learning and growth perspective** (future view), the mission is to achieve the vision by sustaining the innovation and changing capabilities through a continuous improvement and preparation for the future challenges [MDT99]. In each of this perspective, the objectives, the measures, the targets and the initiatives must be presented [KN96].

The cause-and-effect-relationship

The cause-and-effect-relationship helps to leverage the scorecard as a management instrument, therefore the BSC is enhanced with cause-and-effect relationships among the measures from each perspective. It links the strategy to the operational measurement via a cause-and-effect-chain to keep financial balance. The cause-and-effect relationship therefore links the objectives of the different perspectives. E.g., it helps to improve the quality of process (which is located in the ”internal business process” perspective) by having better qualified employees (which is measured in the ”learning and growth” perspective) [Gad05].

1.3.2 Applying the Balanced Scorecard to IT-Controlling

To make a BSC work, with respect to the extra aspects of IT-Controlling, which is already being used in the organization, less effort for the adaption is required than expected. But introducing a BSC still requires high effort, because it needs deep insight into the corporate processes. In some approaches, the perspectives are augmented (e.g. [MGB02]), and in others, they are replaced by specific ones (e.g. [GH05]) or just kept as given [RK07]. The goal of applying the BSC to work for IT-Controlling is to attain the fusion of business and IT and thus achieve better financial results [GH05]. Therefore, a BSC can only be introduced if employees and managers of all departments are integrated in the process of finding performance indicators, because the long term goals and the strategy of the whole company are relevant.

In [GH05], the concept is applied to IT functions and processes. The perspectives have been matched to an IT organization. The **corporate contribution perspective** evaluates the performance of the IT organization from the view of executive management and matches best the financial perspective. The **customer orientation perspective** evaluates the performance of IT as an internal business users’ point of view and fits exactly to the customer perspective. The **operational excellence perspective** provides performance of the IT processes from the perspective of the IT management (and fits to the internal business process). The **future perspective** shows the readiness for future challenges of the IT organization itself and therefore is derived from the future and growth perspective.

In [MGB02], the traditional perspectives are augmented with the perspectives of employees and suppliers, because these parts are crucial in IT organizations. The productive efficiency mainly depends on qualified, motivated and engaged employees, which should be obtained and kept. Suppliers are more important to an IT organization because IT services are delivered over long periods [MGB02].

Another approach by [Gre00] is to use a cascade of BSC. There IT is seen as an internal service provider. By using a cascade of BSCs, a method for business and IT fusion is provided. To achieve this, an IT development scorecard and an IT operational scorecard are designed and enabled for the strategic IT BSC that links to the business BSC [HG04] [Gre00]. By fusing business and IT, this approach deals with the lack of missing business alignment and therefore supports the IT-Governance. Then IT-Governance can superordinate IT-Supply and IT-Demand by employing the business objectives of the company. By applying this approach, the IT can fully be integrated into the business process.

In [MDT99], the BSC is furthermore seen as a decision tool and the structure is kept as given. Here the BSC is used to measure and evaluate IT activities. It provides a framework which can be adapted to IT application projects as well as the IT department or functional area as a whole. They paid respect to the fact that IT typically is an internal service supplier and that IT projects are part of IT organization. By proposing that all key measures are undertaken on an ongoing basis, managers will know what is happening and why it is happening.

1.3.3 Possible metrics in the BSC for IT-Controlling

The metrics are presented arranged into the classical perspectives of a BSC. Metrics can only be applied after having done the preliminary steps of introducing a BSC (see section 1.3.2). Each metric by oneself is not representative or useful to gain any further information. However, the values of the metrics are an evidence for the performance and target achievement if the interdependence and relationships are considered in the interpretation of the results.

Financial perspective

The financial perspective has to deal with objectives like the strategic alignment, the value delivery and risk management as a main concern for IT-Controlling. The measurement challenge within this area lies in the strategic alignment, because it is an overall-metric. It can be measured by a quick self-assessment of at least ten senior managers, where they have to determine how important a particular governance outcome is and how well it is contributed to meet the outcome for IT-Controlling. This outcome should include the cost-effectiveness of IT, effectiveness of IT for growth, effectiveness of IT for asset utilization and effectiveness of IT for business flexibility. Based on these scores, a weighted performance can be calculated [GH05].

Another popular financial metric is the IT budget, which either is expressed as a percentage of sales turnover or as a percentage of total expenses [MDT99]. Other well-known measures are the percentage of sales volume of new products and the contribution to profit [MGB02]. These metrics are old-fashioned and should be teamed with newer approaches to get a better understanding of the work flow and relations between the numbers that are used in the scorecard. To get further information about the cost control beyond the IT budget, the allocation of different budget items, like the IT expenses per employee, can be regarded [MGB02].

In detail, value is a much broader concept than benefits. Value is generated by providing more or faster information to an employee or through accelerating the workflow. It can be measured indirectly (see section 1.3.1) in the marketing and sales performance. Values mostly imply risk, thus the scoring technique attributes value and risk categories to certain scores. Possible measures for risk are: (unsuccessful) business/IT strategy risk, definitional uncertainty (low degree of project specification), technological risk (hardware and software), development risk (inability to put the pieces together), operational risk (resistance to change) and human / computer interface difficulties (IT service delivery risk) [MDT99].

Another useful measurement is the project portfolio control. It focuses on the costs of time to market to spend costs and resources on projects where the costs of time to market are the best. The same pattern is applied when benchmarking project proposal as risk assessment filter. Here the projects are filtered out that have a higher degree of risk in the project estimate [Vog08].

Customers

An IT end-user may be an internal customer or a customer in other companies. Objectives like attracting new customers and satisfying existing customers are common objectives in that perspective, where satisfying is more important than building up market share or acquiring new customers. Therefore, it is critical to monitor customer satisfaction. The company should focus on being the preferred supplier, establishing and maintaining a relationship with the users and satisfying end-users needs [MDT99]. The following metrics can be used to gain information about these objectives: The product and service properties, the customer relationship, the image and reputation, the project status from the customers' point of view, the lost offers, the customer satisfaction, the customer loyalty, the customer profitability, the share of the market, the major project scores and the achievement of the targeted unit costs to get competitive costs [MGB02] [GH05].

Some of these metrics are hard to measure by hard facts and cannot be generated out of the given data in the information system. To get a qualitatively satisfying overview about the current state of the customers' perspective, a broad cross-section of end-users should be surveyed periodically using quantitative methods. Additionally, semi-structured interviews can be done for deeper insights. The results from the surveys should be treated with care, because these are subjective measures, in contrast to the other objective measures that should be a part of a BSC, because these are generated out of system usage data [MDT99].

Internal Business Processes

The objectives concerning the internal business process deal with development and operational processes and the process maturity [GH05]. A goal should be to deliver high quality services to the users at the lowest possible cost, therefore the processes should be managed in a cost efficient manner. Efficiency should be monitored by time but also compared to averages of the industry standards [MDT99]. These can be reached by applying an effective change management.

To validate the quality of processes, the ”Capability Maturity Model Integration” (CMMI) can be used. It helps to optimize processes by defining a concrete target for improvement and linking the organization’s activities with the business objectives. By providing a deeper insight into the processes, processes can be understood, improved and analyzed. The performance of processes can be improved by defining more effective processes. Also the CMMI provides the monitoring and controlling of processes to initiate corrective actions when necessary [MBC03].

To keep processes straight and clean, especially to reach quality services, the complexity of the applications can be taken into account to measure. But here a lack of reliable size and complexity metrics has contributed to difficulties in setting and adhering to project budgets and schedules [MDT99]. The used metrics deal with operational issues and have less impact on the strategy. Metrics like function points are useful to overcome these difficulties and enable the evaluation of software programming productivity. This metric measures the software size based on structured evaluation of user requirements and is independent of language or development methodology and tools. It measures the number of inputs, outputs, inquiries and files used in an application [MDT99]. This metric helps to normalize and therefore provides the possibility to compare quality of services [Vog08].

Another metric regarding quality is the Service Level. The Service Level is a method to achieve transparency about on which level the service is delivered to the inquirer. Aspects that are taken into account for the levels are quality and time. This helps to deal with the available maintenance capacity over the different elements and is often neglected in comparing the amount of staff needed to support and maintain different parts of applications in the portfolio. The costs differ depending on the needed service level. To handle the metric, costs for maintenance and support must be weighted to the cost of an application becoming unavailable. But the usage of metrics of that kind needs discipline: costs and working time must be registered throughout the entire process [Vog08]. Furthermore, the company must know the demand for the different services and on which level these services can be provided. Knowing this the company has a better position to decide what services to provide and what resources are needed to meet particular levels of the service demand [MDT99].

Learning and Growth

The ability of IT to deliver quality services and to lead new technology assimilation efforts in the future will depend on the preparations that are made today and tomorrow. In-house specialists should be provided by continually enhancing the skills of IT personnel to take advantage of technological advances and to gain a thorough understanding of emerging technologies. With innovation and learning efforts, the competence levels can raise and then improve business performance in the future.

The objective of the Learning and Growth perspective deals with the future readiness of the IT business. Since the skills of IT specialists must be focused on preparing them for potential changes and challenges in the future, the need emerges to provide metrics for measuring the IT specialist capabilities with regard to future developments.

To measure the IT specialist capabilities, the following metrics are appropriate: IT training and development budget as a percentage of the overall IT budget, expertise with specific existing technologies, expertise with specific emerging technologies, age distribution of IT staff, perceived satisfaction of IT employees. Metrics that measure the satisfaction and future readiness of employees are: participation in the employee magazine, ideas depending on further educational courses, number or growth of suggestions for improvement, volume of bonuses, certificates, team ability, work experience, period of employment, satisfaction of employees, productivity, labor turnover rate and volume of external service days. To measure the technological part of future readiness, the age distribution of portfolios, platform distribution, technical performance and users' satisfaction with applications can be taken into account. To measure the research into emerging technologies, the IT research budget can be used as a percentage of the overall IT budget. The perceived satisfaction of top management with the reporting on how specific emerging technologies may or may not be applicable to the company is useful as a metric too [MDT99] [MGB02].

1.4 Discussion

Numerous approaches like to see BSCs as a perfect solution for utilizing the power of IT to generate benefits, rather than just looking at it as a required part to do business as usual. But people have to deal with the greater importance of IT for the business process. However, it is rarely mentioned that the BSC is only an instrument to measure performance. It cannot provide a real alignment mechanism[HG05]. It still must be regarded that the BSC is a tool.

The designing and implementing of the BSC must be done with much care. The formulated objectives that are transferred into the perspectives during the design process must be chosen carefully, because they will be realized in the whole BSC and are therefore important for all areas of the company and not only for the IT-units. It is important but difficult to get the right metrics and the right perspectives of each company into the BSC. A wrongly used BSC might lead to crucial misinterpretation and less benefit than an ordinary small performance measurement system would do. Therefore, by defining metrics for the different perspectives of the BSC, it has to be checked whether the metric is covering the operational measures of the objectives and which assumptions are implicitly included in the choice of metric.

Furthermore, it has to be checked if it is possible to generate the values out of the given information system. The values have to be adapted to the given data and these changes have to be appropriate for the organization.

1.5 Conclusion

The BSC is an instrument to support the IT-Controlling process. It helps an organization to change their focus of IT-Controlling from operational views to strategical alignment. The BSC is not only an instrument for the IT division of the company, but also for the entire core functions. It provides a future oriented view on different metrics by applying additional perspectives to the financial (past-oriented) view. It also shows dependencies and relationships by applying the overview. This way, the alignment of the IT-Controlling with the overall business strategy is achieved. The BSC can help to change the view on IT of the managers of the company from costs to benefit. However, it is important that the right metrics and the right perspectives of each company are included into the BSC.

1.6 Outlook

Concerning IT and IT-Controlling, different variants of the BSC are already designed and implemented ([MGB02] [MDT99] [HG05]), where the basic ideas are already provided. With focus on the perspectives and the different kinds of metrics, it can be stated that the process is not finished. It is hard to measure IT with its processes, projects and applications, because of the heterogeneity in this sector. To solve these aspects and to continue the process of involving the IT more and more into the strategical alignment of the business, IT-Controlling and management must get closer. The management has to define the objectives, the controlling has to coordinate and measure the achievement and the IT has to meet the expectation of the business. To accomplish a proper alignment, an intermediate role should be included into the process of implementing the BSC and defining metrics. This role should neither be part of the

IT department nor of the controlling, but should know both departments with deep insight. It could provide benefit for all involved units. Because of the deep knowledge of both parties, the best fitting metrics can be applied without disregarding the IT needs. By fully providing IT needs and using the right metrics, the BSC will work well for the purposes of business. But this approach adds more complexity to the process and thus must be regarded carefully. It must be weighted how much benefit will be generated by applying an additional role.

Bibliography

[Gad05] A. Gadatsch. *IT-Controlling realisieren, Praxiswissen für IT-Controller, CIOs und IT-Verantwortliche*. Vieweg, 2005.

[GH05] W. Van Grembergen and S. De Haes. Measuring and improving it governance through the balanced scorecard. *Information Systems ControlJournal*, 2005.

[Gre00] W. Van Grembergen. The balanced scorecard and it governance. IGI Publishing, 2000.

[HG04] S. De Haes and W. Van Grembergen. It governance and its mechanisms. *Information Systems Control Journal*, 2004.

[HG05] S. De Haes and W. Van Grembergen. It governance structures, processes and relational mechanisms: Achieving it/business alignment in a major belgian financial group. IEEE Computer Society, 2005.

[K0̈5] M. Kütz. *IT-controlling für die Praxis: Konzeption und Methoden*. dpunkt.verlag, 2005.

[KN96] R. S. Kaplan and D. P. Norton. Using the balanced scorecard as a strategic management system. *Harvard Business Review*, 1996.

[MBC03] S. Shrum M. B. Chrissis, M. Konrad. *CMMI: guidelines for process integration and product improvement*. Addison-Wesley Longman, 2003.

[MDT99] M. Martinsons, R. Davison, and D. Tse. The balanced scorecard: a foundation for the strategic management of information systems. *Decision Support Systems*, 25(1), 1999.

[MGB02] R. Blomer M. G. Bernhard. *Report. Balanced Scorecard in der IT. Praxisbeispiele - Methoden - Umsetzung*. Symposion, 2002.

[RK07] H. Schröder R. Kersten, A. Müller. *IT-Controlling, Messung und Steuerung des Wertbeitrags der IT*. Vahlen, 2007.

[Vog08] F W. Vogelezang. Portfolio control — when the numbers really count. Springer-Verlag, 2008.

Chapter 2

Software Process Improvement

Daniel Heidchen

Contents

Abstract: The software development process is one of the major components of software development. However, the success of a software project hardly depends on the develpoment process and how these development process should be imtroduced and optimized. This is the task of SPI. In recent years, this subject has already been carried out in many investigations. A selection of these investigations will be presented here.

2.1 Introduction

Software projects can be influenced by a large number of risks. Some of them are overtime, overrun the budget, misunderstand the customers wish, pursue the wrong goals or totally fail without achieving anything. Software process improvement (SPI) is performed by many organizations to enhance the process of software creation by using systematic and structured methods and models. Without a thorough investigation of the reasons for success of SPI, the most discussions and opinions are based on experience or other subjective know-how. All new ideas and discussions about SPI are useless if they aren't base on provable quantitative models. So the measurement becomes an important part of the research to approve new approaches.

2.1.1 CMMI

The Capability Maturity Model Integration (CMMI) is an important approach in software process improvement. It was developed by the Software Engineering Institute (SEI). CMMI defines needs and requirements for a SPI and it provides organizations with approved practices, to increase their performance in processes at development, services or acquisition.

For each of the scopes of development, acquisition and services one CMMI model is defined (CMMI-DEV, CMMI-ACQ, CMMISVC). It also provides a so called appraisal, that allows the organization to determine their abilities. That means it can be determined how strong the organization processes belong to the CMMI best practices. For appraisal purpose the CMMI defines two approaches. The first approach is the capability level. With this level each process area can be rated. The second approach is the maturity level. This approach appraises the process maturity of the organization.

The deficiency of CMMI is that it does not define any guidance as to implement the various practices. But there are others that describe the detailed improvement of SPI.

In the past the researchers were concentrated on the software processes that they wanted to improve. They described methods how to analyze the existing software processes and give advice what to change. This kind of SPI approaches is characterized with the ”what”. Nowadays the researchers discovered that this is not enough in most cases, so some new approaches were made that concentrate on the ”how”. That means how to implement the process improvement.

In the following sections two models are presented that guide the SPI process.

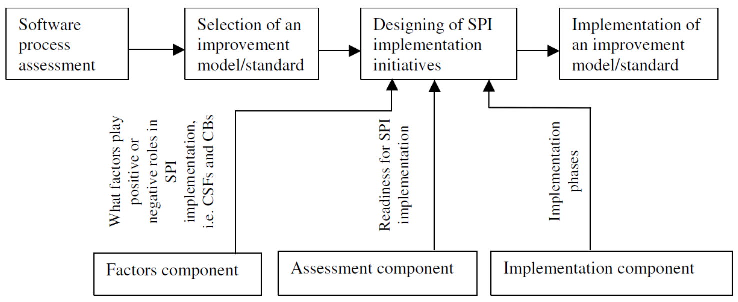
2.1.2 IDEAL

The IDEAL model is an approach to implement a software process improvement. It was designed by Software Engineering Institute (SEI) which also created CMM/CMMI. The purpose of IDEAL model is to instruct the practitioner how to introduce a new software process improvement. The IDEAL model bases on phases consisting of activities that are listed below:

- Initiating: Stimulus for Improvement. Set Context and Establish Sponsorship.
- Diagnosing: Establish Improvement Infrastructure. Appraise current practice. Develop recommendations and document results.
- Establishing: Set Strategy and priorities. Establish action teams, plan actions.
- Acting: Define processes and measure. Plan execute pilots. Plan, execute and track installation.
- Learning: Revise organizational approach. Document and analyze lesson.

2.1.3 SPI Implementation Framework (SPI-IF)

M. Niazi et al. constructed a SPI implementation framework (SPI-IF), which supports practitioners in implementing a SPI. This SPI-IF is a phase-by-phase approach [NWZ05]. In the different phases are different steps carried out to find the appropriate models and methods that can be implemented 2.1. There for the so called factor component analysis the critical success factors and critical barriers. At next, the assessment component checks if the requirements for SPI are given. At least, the implementation component develops a model to assist the practitioner by the SPI implementation. The assessments are based on key factors and the method, which is used to identify these will be discussed in a further section.



Software process assessment
Selection of an improvement model/standard
Designing of SPI implementation initiatives
Implementation of an improvement model/standard
What factors play positive or negative roles in SPI implementation, i.e. CSFs and CBs
Readiness for SPI implementation
Implementation phases
Factors component
Assessment component
Implementation component

Figure 2.1: Software process improvement implementation framework

2.1.4 Overview

In the next sections, different approaches are described which investigate the success of SPI. All these approaches are based on the evaluations of interviews. But they differ in the nature of the methods used for evaluation. Then a discussion of the approaches will take place.

2.2 Success Measurement Approaches

In the literature some different approaches can be found that try to measure the success of a software process improvement. The approaches presented here are mostly based on key factors.

2.2.1 An Empirical Investigation of the Key Factors for Success in Software Process Improvement

Tore Dyba discussed in his paper a full statistical analysis of an empirical survey of a model that predicts the success of software process improvement [Dyb05]. This model mainly depends on 6 independent variables: the key factors for success. The dependent variable ”SPI Success” is the average of the ”perceived level of success” and the ”organizational performance”. There are two moderating varibales: The environmental conditions and the organizational size. An overview of the model is shown in the figure 2.2.

It follows a detailed description of the independent variables:

- Business Orientation: The SPI program must be well defined and in the alignment to the business objectives.

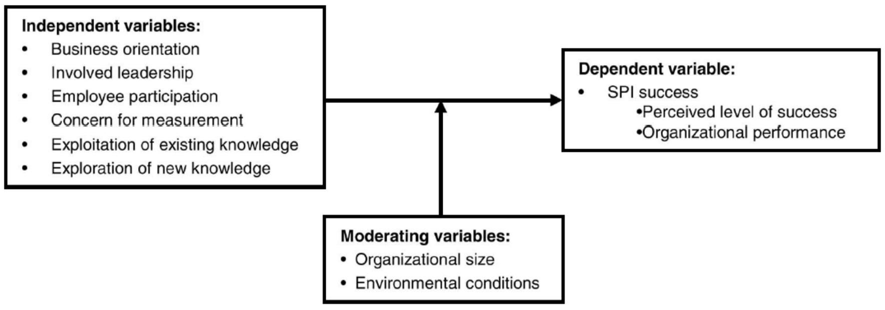

Independent variables:
• Business orientation
• Involved leadership
• Employee participation
• Concern for measurement
• Exploitation of existing knowledge
• Exploration of new knowledge
Dependent variable:
• SPI success
•Perceived level of success
•Organizational performance
Moderating variables:
• Organizational size
• Environmental conditions

Figure 2.2: Model of the dependences

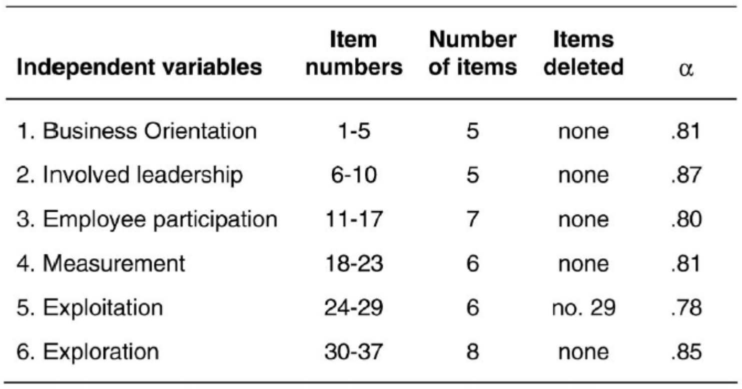
- Involved Leadership: Leadership is required to create a "vision" and to be responsible.
- Employee Participation: Employees must also accept the importance of the SPI and be familiar with the methods.
- Concern for measurement: Measurement means the understand the model and also to have the capability to control, monitor and predict its behavior.
- Exploitation of existing knowledge (Learning Strategy): With each completed project, the organization gets more experience.
- Exploration of new knowledge (Learning Strategy): To achieve benefit for advantages of innovations its necessary to produce new knowledge by research or training.

Reliability and Validation

To test the reliability of the data the Cronbach's alpha [Cro51] method was used. This coefficient is a measure used in the multivariate statistic to describe how well a set of variables measures a single latent construct. Good reliability is achieved by a α-value of above 0.7. The figure 2.3 show that all values are over 0.78. So a good reliability is assumed.

Relationships between variables

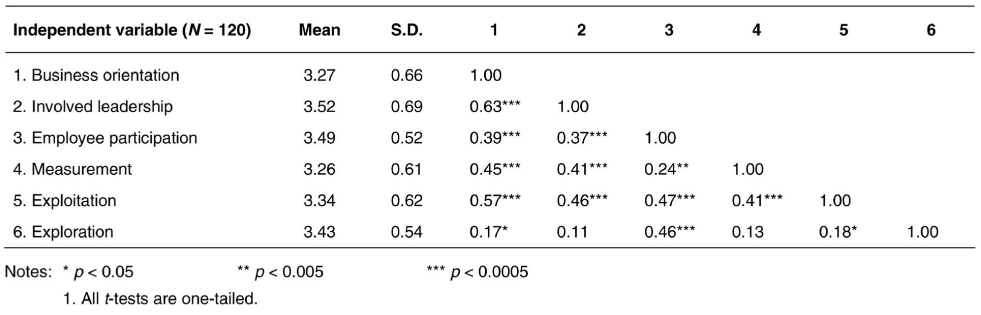
To answer the question of whether there is a correlation between the independent variables, bivarianet correlations between each of them were calculated. The results are shown in figure 2.4. All the variables are highly correlated, except for the exploitation of knowledge with the business orientation, because a



Independent variables	Item numbers	Number of items	Items deleted	α
1. Business Orientation	1-5	5	none	.81
2. Involved leadership	6-10	5	none	.87
3. Employee participation	11-17	7	none	.80
4. Measurement	18-23	6	none	.81
5. Exploitation	24-29	6	no. 29	.78
6. Exploration	30-37	8	none	.85

Figure 2.3: Reliability Analysis

correlation coefficient of 0.17, with $p < 0,05$ doesn't predict good dependence of each other.

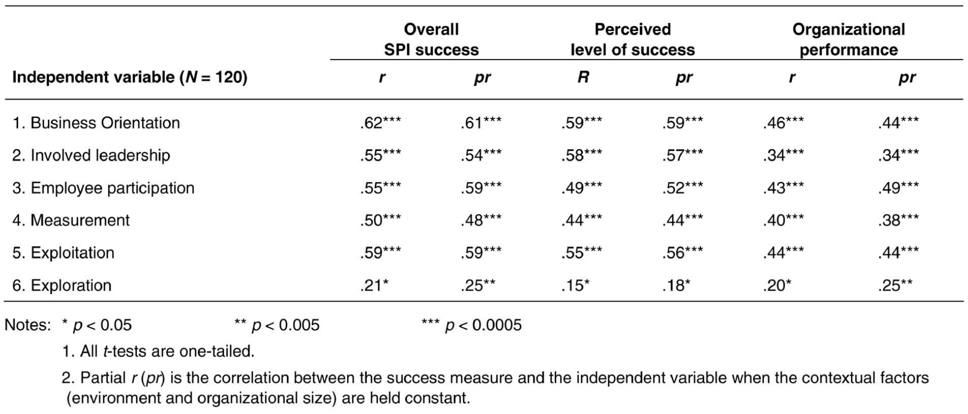


Independent variable (*N* = 120)	Mean	S.D.	1	2	3	4	5	6
1. Business orientation	3.27	0.66	1.00					
2. Involved leadership	3.52	0.69	0.63***	1.00				
3. Employee participation	3.49	0.52	0.39***	0.37***	1.00			
4. Measurement	3.26	0.61	0.45***	0.41***	0.24**	1.00		
5. Exploitation	3.34	0.62	0.57***	0.46***	0.47***	0.41***	1.00	
6. Exploration	3.43	0.54	0.17*	0.11	0.46***	0.13	0.18*	1.00

Notes: * $p < 0.05$ ** $p < 0.005$ *** $p < 0.0005$
1. All *t*-tests are one-tailed.

Figure 2.4: Individual Relationships

The next question was whether there is a connection between each key factor and the success of SPI. For each of the six key factors, the hypothesis is established: *SPI success is positively associated with the key factor*. Again zero-order bivariante correlations (r) are calculated to prove the hypothesis. Also partial correlations (pr) are used to show the influence of the contextual vaiables (which are environment and organization size). The correlation coefficients r and p can assume values in the range from -1 to 1. The larger the absolute value is, the stronger the linieare correlation is. The results are shown in figure 2.5. So the factors have a good correlation with the success. Neither the results of the zero-order correlation nor the results of the partial correlation to have a large variety in their values. Thus, the bivariante correlation tests support the six hypothesis.



	Overall SPI success		Perceived level of success		Organizational performance	
Independent variable (*N* = 120)	*r*	*pr*	*R*	*pr*	*r*	*pr*
1. Business Orientation	.62***	.61***	.59***	.59***	.46***	.44***
2. Involved leadership	.55***	.54***	.58***	.57***	.34***	.34***
3. Employee participation	.55***	.59***	.49***	.52***	.43***	.49***
4. Measurement	.50***	.48***	.44***	.44***	.40***	.38***
5. Exploitation	.59***	.59***	.55***	.56***	.44***	.44***
6. Exploration	.21*	.25**	.15*	.18*	.20*	.25**

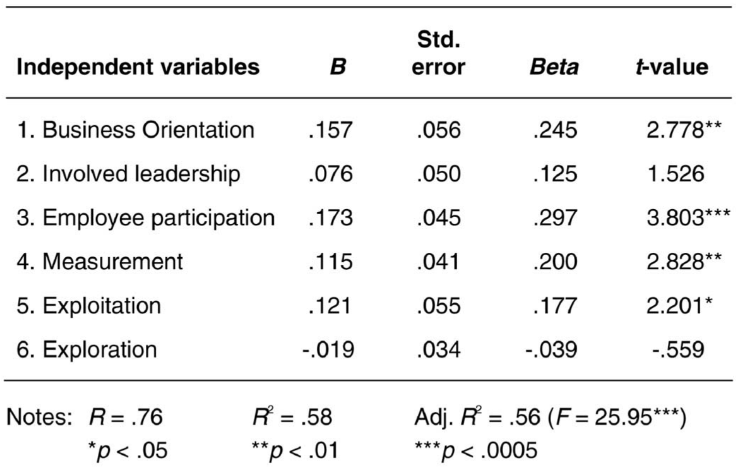
Notes: * $p < 0.05$ ** $p < 0.005$ *** $p < 0.0005$

1. All *t*-tests are one-tailed.
2. Partial *r* (*pr*) is the correlation between the success measure and the independent variable when the contextual factors (environment and organizational size) are held constant.

Figure 2.5: Test Hypotheses

Overall Relationships

Cohen's effective size is defined as $f^2 = \frac{R^2}{1-R^2}$ where (R^2) is the squared multiple correlation [Coh88]. Some transformations according to Fisher r-to-Z are necessary [Kon88]. The results of the test statistic are show in figure 2.6.



Independent variables	*B*	Std. error	*Beta*	*t*-value
1. Business Orientation	.157	.056	.245	2.778**
2. Involved leadership	.076	.050	.125	1.526
3. Employee participation	.173	.045	.297	3.803***
4. Measurement	.115	.041	.200	2.828**
5. Exploitation	.121	.055	.177	2.201*
6. Exploration	-.019	.034	-.039	-.559

Notes: $R = .76$ $R^2 = .58$ Adj. $R^2 = .56$ ($F = 25.95$***)
*$p < .05$ **$p < .01$ ***$p < .0005$

Figure 2.6: Overall Relationship

The analysis show that the SPI success has a positively associated orientation with the a key factors Business Orientation, Employee Participation, Concern for measurement, and Exploitation of existing knowledge, because statistical test results of B are higher than 0.51.

The analysis revealed that the key factors "involved leadership" and "exploration of new knowledge" did not have a positively associated orientation with the SPI success, because statistical test results of B are lower than 0.51.

In summary one can say that the analysis of the factors was very detailed. But there was no procedure specified with which one can conduct an audit of a SPI initiative.

2.2.2 A framework for assisting the design of effective software process improvement implementation strategies

The SPI-IF, that was developed by Miazzi see. Section 2.1.3 is based on key factors. For this research a survey was taken to collect some data for an empirical analysis. First, literature was searched to identify the key factors that are described by other researchers. This approach was objective, but to reduce the researcher's bias this work was taken by a person that wasn't familiar with this topic. Secondly twenty-nine software organizations were examined. Then the transcripts of the interviews were used to identify major themes. Than the themes were grouped into categories. If one of the categories was mentioned in more than 30 percent of the interviews and has a positive alignment with the SPI success it becomes a critical success factor. Negative aligning categories become critical boundaries.

The key factors are divided in the categories: critical success factors (CSF) and critical bariers. The CSF that were found during the categorization process are: Senior management commitment, SPI awareness, staff involvement, experienced staff, defined SPI implementation methodology, reviews, training and mentoring, staff time and resources, creating process and action teams/external agents. The CB that were found during the categorization process are: organizational politics, lack of support, lack of resources, inexperienced staff/lack of knowledge and time pressure.

Implementation maturity model

Based on this CSF and CB an implementation maturity model (IMM) is developed to assess the maturity state of the SPI implementation. The IMM is an adaption of the CMMI. It's structure bases on the three dimensions: Maturity stage dimension, CSF & CB dimensions and Assessment dimension.

The IMM maturity stage dimension defines four levels see figure 2.7. This levels are adoptions of maturity levels of the CMMI. Initially the first level is directly assigned without and adaption. At the second level comes the awareness. In 59% of the critical success factors the awareness is mentioned as one of the important parts of SPI. It is clear the all practitioner should be fully understand the benefits of SPI. Level 3 and 4 are emerged form the CMMI levels 3 and 5. The residual level 2 and 4 of CMMI are not adapted.

At CMMI for each maturity level a set of process areas (PA) is assigned. In the

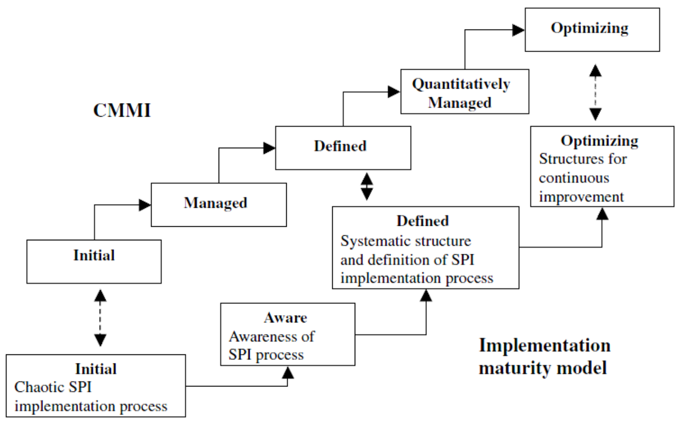

CMMI
Initial
Managed
Defined
Quantitatively Managed
Optimizing
Initial Chaotic SPI implementation process
Aware Awareness of SPI process
Defined Systematic structure and definition of SPI implementation process
Optimizing Structures for continuous improvement
Implementation maturity model

Figure 2.7: Maturity stage dimensions

IMM model this PAs are replaced by CSFs. So groups of CSFs and CBs are build and attached to the maturity stages.

- Awareness: Senior management commitment, training and mentoring, staff involvement, awareness of SPI
- Organizational: Creating process action teams, experienced staff, staff time and resources, formal methodology
- Support: Reviews

Finally, the assessment is carried out. In this dimension each CSFs and CBs are evaluated to assess maturity stage. All practices of a CSF or a CB are evaluated with an assessment instrument developed by Motorola [Das94].

- The three-dimensional score of the assessment is calculated.
- The components of the score are summed up and divided by three.
- Calculate average of all practices. That is the score of the CSF.
- If the CSF or CB score is higher than 7 the factor has been correct implemented.
- To reach a certain maturity level all CSF and CB must have an average score higher than 7.

This approach to predict SPI success is based on a maturity model. In alignment with the maturity stage the quality of the improvement process increases and the risk sinks.

2.2.3 Modeling the Likelihood of Software Process Improvement: An Exploratory Study

El-Emam, et al. had conducted a re-analysis of two studies to examine the influence of specific factors on the success of SPI [EEGJJ01]. These studies were limited to the bivariant analysis of key factors. Interactions between these factors were not considered. In his study, a multivariate analysis was carried out, to develop a model, that can predict the success of a SPI effort.

Model

The model assumes that there are two classes of independent variables 2.8. The first class contains fourteen organizational factors (ORG[i]). These factors describe the organization in which the SPI takes place. The response categories of the questions of these factors were ”Substantial”, ”Moderate”, ”Some” and ”Littlie if Any”. In the second class four process factors (PROC[i]) were comprised. Process factors refer directly to the examined SPI process and has the response categories of ”yes” and ”no”.

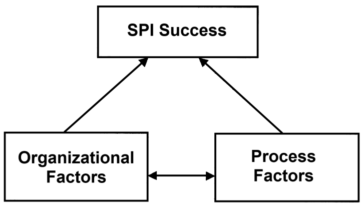


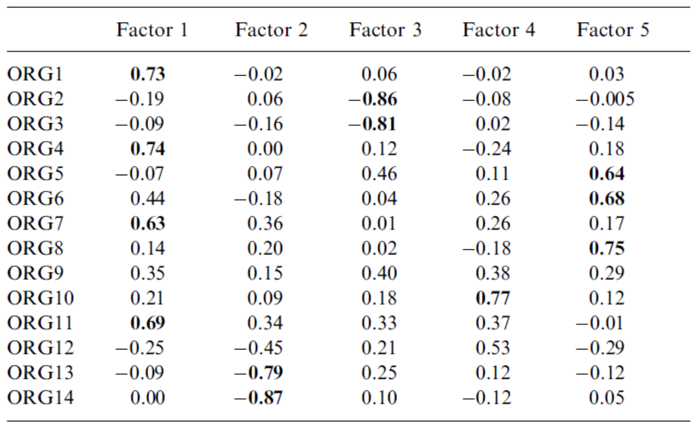
Figure 2.8: Model of SPI success and factors

Principal components analysis (PCA)

One assumption of this study was that some of these variables are measuring the same entity. For this analysis, principal component analysis (PCA) was used [KM78]. It is a technique of the multivariate statistics that simplify large data sets by reducing many statically variables to few linear combination. In other words, the method combines factors that have a value higher than a special cutoff-value to a new one. A value of 0.63 is chosen as a cutoff point [Com73]. The cutoff emerge groups that are marked on the figure 2.9.

This emerged factors are summed up and interpreted as composite variables:

- Commitment: This factor measures how strong the SPI is funded and how much the management is involved.



	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
ORG1	**0.73**	−0.02	0.06	−0.02	0.03
ORG2	−0.19	0.06	**−0.86**	−0.08	−0.005
ORG3	−0.09	−0.16	**−0.81**	0.02	−0.14
ORG4	**0.74**	0.00	0.12	−0.24	0.18
ORG5	−0.07	0.07	0.46	0.11	**0.64**
ORG6	0.44	−0.18	0.04	0.26	**0.68**
ORG7	**0.63**	0.36	0.01	0.26	0.17
ORG8	0.14	0.20	0.02	−0.18	**0.75**
ORG9	0.35	0.15	0.40	0.38	0.29
ORG10	0.21	0.09	0.18	**0.77**	0.12
ORG11	**0.69**	0.34	0.33	0.37	−0.01
ORG12	−0.25	−0.45	0.21	0.53	−0.29
ORG13	−0.09	**−0.79**	0.25	0.12	−0.12
ORG14	0.00	**−0.87**	0.10	−0.12	0.05

Figure 2.9: Principal components analysts results

- Turnover: The extent of this variable describes the turnover of the middle management and the technical levels.
- Politics: All politically motivated activities that inhibit the SPI process are related to this factor.
- Respect: This factor show how much the people who are involved in the SPI initiative, are respected for their technical skills.
- Focus: The focus factor describes nearly the same as the turnover factor but not on the employee level but rather on the senior management level.

Classification and Regression Trees (CART)

In this subsection, the probability of the success of a SPI initiative will be derived with help of a ”classification and regression tree”. The classification and regression tree is a method to create a binary tree that can be used to classify data or to derive a probability of data. But the correctness of the results of the classification or the derivation depends on an propability. To construct the tree the CART algorithm is used [BFOS84]. The algorithm uses the variables from the previous section for the nodes of the tree. At the beginning of the algorithm a large tree is created and then it is pruned from the bottom up. A splitting algorithm is used to decide on which independent variable to split and where to split. For the splitting criterion the Gini measure was used [BFOS84]. The resulting tree is shown in figure 2.10.

The assessment of the success of a SPI initiative can be derived from this tree. We giving an example: first of all die factors ”focus”, ”commitment” and ”politics” should be assessed by an expert in SPI. Than the tree of traversed form the root to the terminal nodes depending on the conditions on each node.

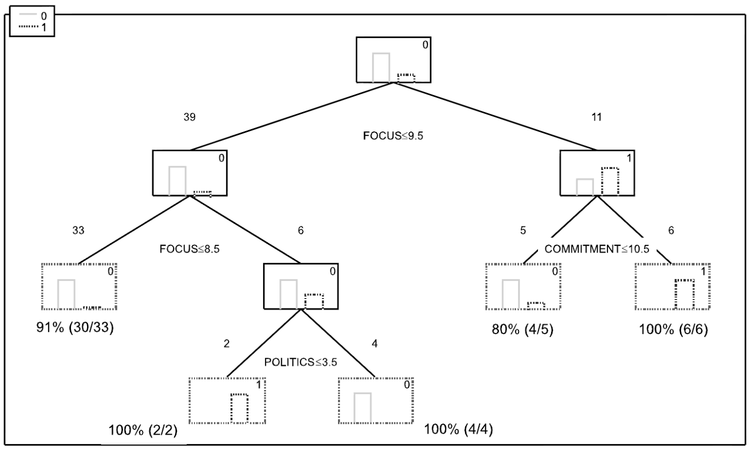


Figure 2.10: Classification and Regression Trees (CART) result

At last the prediction of success is given by the reached terminal node. A terminal node that has the values $(1, 100\%)$ predict a success of 100%. However, it must be noted that in our case the probability to get the correct classification with this model is 50% and 76%.

2.2.4 A framework for evaluation and prediction of software process improvement success

In this study Wilson D. et al. constructed a framework to predict the success of a SPI initiative. This framework bases on a framework [JB93], which was designed to measure a software metric program.

For the adaptation, the framework's four perspectives have been adjusted. Each perspective were determined by factors which were defined by question. The perspectives and their scope are listed below:

- Context (C): business goals and senior management.
- Input (I): resources, persons respect and number of people.
- - Process motivation (PM): promotion of the SPI initiative.
  - Process responsibility (PR): responsibility and independence of the SPI team.
  - Process improvement (PI): initialization of SPI determination of objective.

  - Process training (PT): training in SPI, awareness of the business processes.

- Products (P): Results and feedback.

For all these perspectives questions were prepared that will be rated within a survey. The responses has been divided according as whether the software organization has success or unsuccess in the SPI implementation. The calculated average of a response is assigned to a factor. After this, factors have been examined concerning their significance. These factors were determined, which values had a big difference between the response of the successful companies and the unsuccessful companies. Then the Tuky test [Tuk59] was conducted to prove if there is a significance difference. For these significant factors, hypotheses were made and investigated regarding their particular importance for the success.

As the result, four critical factors have been found: (C6) Was senior management commitment available? (I2) Was SPI program staffed by highly respected people? (PI1) Were the important initial processes to be improved defined? (PI3) Were capabilities provided for users to explain events and phenomena associated with the program?

2.3 Discussion

In this discussion, the four different publications were presents, which investigated the success of SPI. On the development of the various models presented in this paper correct statistical methods were used. But the input data for this were collected by interpreting some means of subjective questions. Thus, a special term may not only have different meanings, but also be assigned different strength. We must keep watch that the subjective bias remains within the tolerances. Significant advantages and disadvantages will be show.

2.3.1 Compare of the Factors

All approaches that has been examined, explore the success of SPI in dependence of the key factors. The key factors that were found mostly describe the same properties: Senior management commitment, experience of the staff, clear and structured goals. An advantage of all this approaches is that they reduces the critical success factors strongly and a clear number of them remains.

2.3.2 Compare of the Methods

The strategy of all the approaches was it to carry out a survey on software companies which have experience with SPI. Most of the approaches use different methods to categorize the factors into groups. Than other methods were used to determine their relationship to the success.

In my opinion the approache of Miazzi in section 2.2.2 has the best method to identify the key factors, because in this approache the garthering is based on the results of the survey. All other methods are defining some key factors previously.

The assessment has also different approaches. In section 2.2.2 the result of the assessment is a mandatury level. The advantages of this are that are clearly defined levels and the properties which were assosiated it. So the assessor can appraise the caracteristics of the SPI initiative. The evaluation described in section 2.2.3 only gives a propability of the success and there are no conclusion about the scopes that have to be improved.

2.4 Conclusion

In this paper four different approaches to determining the success of SPI have been presented. Most of them focused on the identification and analysis of so-called key-factors. To this purpose, surveys were carried out in software organizations. Based on the experience of the software process managers or other responsible persons, they have answered the questions. The evaluation of the success was made on different ways. One uses a mandatory model to qualify the SPI initiative other approaches predict the probability of the success.

Bibliography

[BFOS84] L Breiman, J Friedman, R Olshen, and C Stone. Classifcation and Regression Trees. Wadsworth and Brooks Cole. 1984.

[Coh88] J. Cohen. Statistical Power Analysis for the Behavioral Sciences. *New Jersey: Laurence Erlbaum*, 1988.

[Com73] A Comrey. A First Course on Factor Analysis. *Academic Press*, 1973.

[Cro51] L Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334, 1951.

[Das94] M K Daskalantonakis. Achieving higher SEI levels. *IEEE Software*, 11(4):17–24, 1994.

[Dyb05] Tore Dyba. An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering*, 31(5), 2005.

[EEGJJ01] K. El-Emam, D. Goldenson, McMurley J., and Herbsleb J. Modelling the Likelihood of Software Process Improvement: An Exploratory Study. 2001.

[JB93] R. Jeffery and M. Berry. A framework for evaluation and prediction of metrics program success. *First International Software Metrics Symposiom*, pages 28–38, 1993.

[KM78] J. Kim and C. Mueller. Factor Analysis: Statistical Methods and Practical Issues. 1978.

[Kon88] S. Konishi. Normalizing Transformations of some Statistics in Multivariate Analysis. *Biometrika*, 68(3):647–651, 1988.

[NWZ05] M Niazi, D Wilson, and D Zowghi. A framework for assisting the design of effective software process improvement implementation strategies. *Journal of Systems and Software*, 78(2):204–222, 2005.

[Tuk59] J. Tukey. A quick, compact, tow-sample test to Duckworth's specifications. *Technometrics*, (1):31–48, 1959.

Chapter 3

Metric Maturity Model

Carlos Gomez

Contents

Abstract: Measurement is one of the most important activities in software process improvement because it provides an indicator for the process improvement. But measurement is also a process and it can be and should be also improved. In this document are presented some models from the literature that can help in assessing the maturity of certain process and to improve it.

3.1 Introduction

Software process improvement is one of the most challenging activities within an organization and implies hard work in order to perform such improvements. Some standards and model have been developed to aim in this task e.g the Capability Maturity Model Integrated (CMMI) and SPICE. The appropriated implementation of software process improvements depends on the maturity that an organization possesses for performing this activity. This maturity, as it will be seen later, can also be assessed. Measurement process plays a big role in the improvement of software processes providing a reference to compare the performance between software process implementations. But as name says, Measurement is also a process and has to be introduced into the organizations, and then it should be assessed and improved.
Models like CMMI and SPICE were developed for the assessment and improvement of the software development process, although they also contemplate the measurement process, they do not provide a specific model in order to assess its maturity. Some questions like, what are factors that affect the good implementation of measurements? or how do we assess the measurement process? or how is the measurement improved? are going to be covered within this document.
This document present in the section 2 some concepts related with process maturity in order to provide a background. After that in section 3 maturity models for the software development process are described in order to understand how is build such a model and because other models are based on them. In section 4 the importance of the software process improvement is explained through the presentation of a framework that aims to implement and assess the SPI in the organization. At section 5 two models for the assessment and improvement of the measurement process are shown. Finally in section 6 a conclusion is given where some aspects presented in the document are discussed.

3.2 Concepts related with Process Maturity

It is common to hear about maturity process or capability of a process but what does it really means? In this section are described some concepts related with maturity and capability into an organization context as well as the importance of those concepts in the software processes.

3.2.1 Process Capability, Process Performance and Process maturity

The application of the definitions maturity and capability for software processes has been made by [PWCC95], resulting the following concepts.

Software Process Capability

Capability means the ability to do something. When it is applied to a software process, it is possible to say that the capability of a software process describe the range of expected results that can be achieved by following a software process. The software process capability of an organization provides one means to know which abilities a process possess while developing software, helping to predict the most likely outcomes to be expected from the next software project into the organization.

Software Process Performance

Another concept related to the software process is performance. The performance of a software process is defined by the actual results that have been achieved following this process. The difference between capability and performance of a process is that process performance focus on already achieved results and process capability on results expected.

Software Process Maturity

The maturity of software process is a relation to which a specific process is explicitly defined, managed, measured, controlled and effective. The maturity of a process indicates both the richness of an organization's software process and the consistency with it is applied in projects throughout the organization. The capability of a process indicates the achievement that can be expected applying the process, and the better a process defined, managed measure and controlled is, the more capable it is. Therefore the maturity of a process implies a potential for growth in capability.

3.2.2 Organization Maturity

In order to understand in what consists the maturity of an organization, here are defined the characteristics that distinguish a mature organization from an immature one. A immature organization is the one in which the software processes is normally not defined, everything is doing by improvisation during the course of the project or sometimes even if a software process has been specified, it is not rigorously followed or enforced. These organizations normally exceed schedules and budgets because they estimate just to win the project but with unrealistic deadlines and resources.
In the other hand there exist the mature organization which has better process definition, properly documented, usable, and consistent with the work that actually is being performed. In contrast to an immature organization, a mature organization possesses the ability to manage software development and maintenance. The improvement of the process definitions are made when it is required and through controlled pilot-test and/or cost effective analysis [PWCC95].

3.2.3 Critical Success factors and Critical Barriers

The success factors are defined as the key areas of activity in which favorable results will be necessary in order to reach goals. In other words, success factors are the variables that will most affect the success or failure in the pursuit of a goal [BR81].
There are for every branch in the industry success factors; those have to be identified and the organizations should focus its attention on them because they make the difference between success and failure.
After the identification of the success factors, these can be used to aid in the company's planning process, to enhance communication within the organization's management and to support the system's development.
At the same time in the implementation of a process there are negative factors known as critical barriers. On critical barriers the organization must also focus its attention because they are the principal variables that could stop a well process implementation and therefore it is important to identify them in order to avoid or minimize its effects.

3.2.4 Maturity Model

With the intention to improve the development implementation process, some maturity models have been developed, like CMMI or SPICE. These models are described in section 3. It is important to know what is the purpose of a maturity model in order to understand for what it can be used and to avoid

some misunderstandings.
A maturity model has to be seen as a tool that help to assess the maturity of a process and to improve it through a well define series of best practices that when correctly follow will result in many benefits within the organization processes.
A maturity model normally presents the activities that have to be made in order to improve a process but do not show how to implement them. Therefore when an organization adopts a maturity model it means not that its productivity will be double instantaneously, a maturity model require hard work in order to be implemented.

3.3 Known Maturity models for the development process

In the field of software development process some maturity models have been developed, one of them is the CMM or the new version CMMI (Capability Maturity Model Integrated). This model was developed and presented in 1991 by the Software Engineer Institute (SEI), a research and development center sponsored by U.S. Department. Another model is SPICE (Software Process Improvement and Capability Determination) which was developed by ISO and after the CMM but based on the experiences of it. In this section an overview of these two models is given.

3.3.1 CMMI

CMMI for Development consists of best practices that address development and maintenance activities applied to products and services. It is basically a detailed requirement catalogue ordered in five levels. An organization which does not fulfill the criteria of the first level will remain in the level 1. In other case the organization is set to higher level and if the organization fulfills all the criteria of the model, it will be at level 5 [HL07].

Structure

CMMI supports two improvement paths; one is called continuous which enables organizations to incrementally improve processes corresponding to an individual process area selected by the organization. The other path, called staged, enables organizations to improve a set of related processes by incrementally addressing successive sets of process areas. Because of space constraints and because other models later here presented are based on the staged path, it will be just described the structure of the CMMI in staged path.

The CMMI is structured in maturity levels and for every level there exist key process areas corresponding to that level. Every process area is described through goals that should be achieved and for each goal practices are defined to help in the achievement of the goal. CMMI differentiate between specific goals, which are for a particular process area and general goals which apply for all of the key process areas. The same differentiation is made by the practices. The levels are the following:

Level 1 - *Initial*: Process is ad hoc and chaotic. Success in the organization depends on the personal effort of people and not in proven process.
Level 2 - *Managed*: In this level a manage process is performed. It means, the process is planned, monitored and reviewed. The disciplines reflected on this level help to ensure that existing practices are retained along the all process.
Level 3 - *Defined*: at this level a defined process is established, in which a standard process is defined, documented and used. This process will be use for every project within the organization.
Level 4 - *Quantitatively Managed*: In this level the defined process from level 3 is controlled using statistical and other quantification techniques. Quality and process performance is understood in statistical terms and is managed throughout the life of the process.
Level 5 - *Optimizing*: in this level a quantitatively managed process is improved based on an understanding of the common causes of variation inherent in the process.

Assessment

The CMMI assessment approaches are the following:

Official Review (Appraisal). It is when an organization is been reviewed by an official CMMI appraiser. The appraiser assesses in the organization each process area defined by the CMMI and depending on the criteria fulfilled by the organization, it is set to the corresponding maturity level. An official certificate is issue by the appraiser after the assessment is completed. This certificate certifies the maturity level that the organization possesses.
Internal Review (Assessment). This approach is use in order to identify weaknesses into processes allowing the organization to improve his processes.

3.3.2 SPICE

SPICE is a framework for the assessment and improvement of processes. It was developed by the ISO (International Organization for Standardization) and the IEC (International Electrotechnical Commission) and in 2003 became an international standard ISO/IEC 15504.

Structure

SPICE also defined a maturity model like the one from CMMI which contains the following levels:
Level 0 - *Incomplete*: There exists no process.
Level 1 - *Performed*: There exists a process which is used and delivers the needed requirements.
Level 2 - *Managed*: The process is planned, controlled and adapted.
Level 3 - *Established*: There exists a defined and documented process which is use for every the project of the entire organization.
Level 4 - *Predictable*: The process and finished product quality is been determined and analyzed continuously in order to determine if the guidelines were fulfilled.
Level 5 - *Optimizing*: If the process has some issues, those can be identified and corrective actions are implemented.

Assessment

The capability of processes is measured using process attributes, those attributes are selected for an official review, the reviewers will evaluate each of this attributes for every of the process that should be assessed. SPICE does not assess all the process at one, but every process will be as a unit assessed and grade within the maturity model. The assessment of each attribute will be evaluated within an ordinal scale which has the values not, partially, largely and fully achieved.

CMMI and SPICE share some common characteristics because SPICE was developed based, among other aspects, in the experiences of CMM. CMMI, which is the last version of the CMM, uses many of the ideas of SPICE. One of the characteristics that distinguish SPICE from CMMI relies on the level of model customization. SPICE allows the organizations to develop its own model (e.g. Automotive SPICE) for the assessment and improvement of processes and CMMI does not allow really customization.

3.4 Maturity Model for Software Process Improvement

The adoption of practices in the organization or most known as a software process improvement (SPI) is one of the most challenging activities into software organizations. As mention at Niazi et al [NWZ05], many advances had been made in this field but nevertheless there is still some issues. One of these issues can be observed in the developed models like CMMI or SPICE, which help in the implementation of software process advising which aspects are necessary to be implemented, but they do not provide an effective strategy to execute the implementation.

3.4.1 Implementation Framework for Effective SPI implementation

Niazi et al. propose a framework that aim in the effective SPI implementation [NWZ05]. This framework was developed making emphasis in what and how to implement activities for a SPI. This framework has three components i.e. SPI Implementation factors, SPI assessment component and SPI implementation component.

1. **SPI implementation factors**
   In order to identify why the ”what” is important, they made a list of factors that are critical in SPI implementation. These are divided in two categories, the positive (CSF’s) and the negative ones (CB’s). The identification of these factors was made through the analysis of the literature and implementing some interviews using the CFS interview procedure developed by Rockart et al. [BR81] . Some of the CFS’s included in the SPI-IF are:
   Senior management commitment, SPI awareness and Staff involvement.
   And some of the CB’s are:
   Organizational politics, lack of support and lack of resources.

2. **SPI implementation component**
   In the implementation of SPI the how to implement is another issue that were found by Niazi et al. and in order to help practitioners in the implementation of SPI an implementation model (SPI-IM) was developed [NWZ03]. The SPI-IM is divided into 6 phases in which different CSF’s and CB’s are contained. Besides that more specific practices were designed for each CSF and CB. The phases guide the practitioner through the implementation of the SPI, the first phase is the *Awareness* phase which is selected as an ongoing phase, that is because SPI is an expensive and long-term approach and in order to get support for improvement of management and practitioners it is very important to promote awareness during the all SPI implementation program. After that the *Learning* phase appears, in this phase the training in SPI skills is emphasized. The next phase is the *pilot implementation* where the practitioners implement SPI programs at low level and observe how successfully it is. Once the pilot implementation was established, the *SPI Implementation action plan* follows as a next phase, here a proper plan is made in which a proper

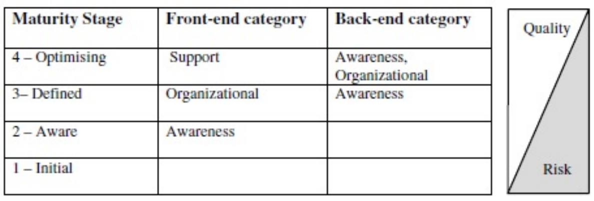
implementation activities, schedule, allocated resources, responsibilities, budget and milestones should be design. The next step in the SPI-IM is the *Implementation across the organization*. Here after the proper planning and the experience collected in the pilot implementation, practitioners start implementing SPI practices in other areas/departments of the organization in order to have a uniform development approach and maturity across the organization. The last phase is called *Maintenance* where a continuously monitoring and support the previously implemented SPI activities is made. Maintenance will also help to improve the already implemented methodology.

3. **SPI assessment component**
   The assessment of SPI implementation maturity can help organizations in the successfully implementation of SPI initiatives. In models like CMMI or ISO 9001 no attention has been paid to the assessment of SPI implementation maturity of organization, therefore in this framework an implementation maturity model (IMM) was developed.

Implementation Maturity Model

The IMM was developed adapting the CMMI perspective. The general structure of the IMM is the 4 evolutionary maturity levels. Each of them containing different CSF's and CB's and for every factor different practices are design to guide the assessment and implementation of each factor.
The maturity levels are:
Level 1 - *Initial*: adopted from CMMI and is the one where the SPI implementation is chaotic.
Level 2 - *Aware*: this level is called aware because the good implementation of SPI initiative will be beneficial only if the practitioners are aware of its benefits; this is the reason why it is important to promote awareness in the very beginning of the SPI implementation.
Level 3 - *Defined*: the SPI implementation processes are documented, standardized and integrated into a standard implementation process for the organization.
Level 4 - *Optimizing*: here is when in the organization structures for the continuous improvement have been established.
The CMMI consists of 22 process areas which are categorized across the five maturity levels and can be split in four categories, i.e. process management, project management, engineering and support. In the IMM instead of the process areas the identified CSFs and CBs are used and analog to CMMI, these are also split in three categories i.e. 'awareness', 'organizational' and 'support'. The division of these categories between the different maturity levels was made using the perception of process areas division among different maturity levels of CMMI. The assignation can be seen in the figure 3.1., where

the front-end category is the current category of each level and the back-end category are the categories of previous levels that should be continuously monitored.



Maturity Stage	Front-end category	Back-end category
4 – Optimising	Support	Awareness, Organizational
3– Defined	Organizational	Awareness
2 – Aware	Awareness	
1 – Initial		

Quality
Risk

Figure 3.1: CFS's Dimension [NWZ05]

Assessment

In the Assessment each of the CSFs and CBs is measured to assess how well the factor has been implemented. A method developed by Daskalantonakis was adapted to make the assessment [Das94]. This adaptation of the Daskalantonakis method evaluates each of the practices defined for the CBFs and CBs in three different dimensions. The first one is called *Approach* which has as criteria the organization commitment and management support for the practice as well as the organization's ability to implement the practice. The second dimension is *Deployment* where the criteria are the breadth and consistency of practice implementation across project areas. And the third one is called *Result* where the breadth and consistency of positive results over time and across project areas are contemplated.

This framework gives us an overview of what is necessary in order to implement a SPI. It is necessary to identify which factors will affect the process implementation in order to know where to focus on. Then it is required a tool or procedure that will allow to guide the implementation telling how should the activities be done in order to do it successfully, but how to know if the SPI implementation was really well implemented? Therefore it is required a tool to assess how good or how bad is the implementation, and this tool is the IMM.

3.5 Maturity Models for Measurement programs

Measurement is one of the key activities in the improvement of software processes because it provides a means to assess if a goal or process is fulfilled. But how to know if a measure program is done in the right way or if it can be improved, models like CMMI or SPICE do not really provide a basic to assess the maturity of a measurement process, therefore in this section some measurement maturity models from the literature are presented.

3.5.1 Critical Success Factors and Critical Barriers in the Measurement programs implementation

As already said the success factors play a big role in the improvement of a process and for measurement processes it is not an exception. A list of success factors for the well implementation of a measurement program was identified by Fenton et al [HF97]. Some of the success factors contained in this list are the following: Incremental implementation, Well-planned metrics framework, Use of existing metrics materials, Measurement process transparent to developers, Usefulness of metric data, Ensure that data is used and seen to be used, Use automated data collection tools, constantly improving the measurement program and provision of training for practitioners.
Other authors identified those barriers or reasons that make that a measurement programs fail, those reasons are: measures not tide to business goals, irrelevant or not understood by key players, perceived to be unfair or resisted, motivated wrong behavior, expensive, cumbersome, no action based on the numbers, no sustained management sponsorship [GH01].
As mention before, the organization should focus on this success factors and barriers in order to improve, in this case, the measurement process. Some of the practices and process areas in the following models take into account the success factors and barriers here presented.

3.5.2 M-CMM

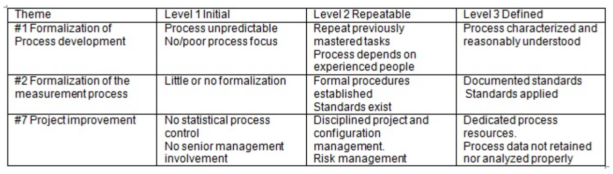
Because of the lacking of a maturity model to assess the maturity of a measurement process Niessink et al. have proposed a Measurement-Capability Maturity Model (M-CMM) [NvV98]. The objectives of this model are to enable organizations to assess their capabilities with respect to software process measurement and to give some directions for the improvement of their measurement capability. They defined measurement capability as ”the extent to which an organization is able to take relevant measures of it products, process and resources in a cost effective way resulting in information needed to reach its business goals”.
This model defined maturity levels similarly to those in the CMMI starting from the Initial level where the organization has no defined measure process, at level 2 (*repeatable*) organization are able to collect some metrics but every project has his own measurement goals, the level 3 (*define*) is when the organization determine a basic set of measurement that each project has to collect and also an organization measurement database is created, in the

level 4 (*managed*) the organization is able to assess the costs of measurement and technology is being used to perform the measurement process efficiently, finally at level 5 (*optimizing*) measurements are constantly monitored and changed when necessary.
For each level a set of key process areas are define as follow:
Level 1 - *Initial*: no key process areas
Level 2 - *Repeatable*: Measurement Design, Measurement Collection, Measurement Analysis and Measurement Feedback.
Level 3 - *Defined*: Organization Measurement Focus, Organization Measurement design, Organization Measurement Database and Training Program.
Level 4 - *Managed*: Measurement Cost Management and Technology Selection
Level 5 - *Optimizing*: Measurement Change Management.
This model with its maturity levels and its key process areas provide organizations with both a measurement scale to assess their measurement capability and directions to future improvement.

3.5.3 Daskalantonakis's method for assessing the software measurement process

Another model for assessing the maturity of a measure process within an organization was developed by Daskalantonakis et al [MKDB90].
This model is based on a series of themes identified by the author that influence the software measurement technology maturity and argued that in order to assess and improve the measurement maturity of an organization it is necessary to understand how these themes affect the measurement process and the relationship between then.
The themes that influence the measurement technology maturity of an organization found by Daskalantonakis et al. are based on assumptions that will help in the definition of the maturity levels, these assumptions and themes are the following:
1. A well-defined, quality-focus, software development process will very likely result in a qualitative software project and product. Under this assumption the following theme is important: Theme 1: Formalization of the development process.
2. Measurement is facilitated by, and facilitates a well-defined, software development product. Therefore the following themes are important: Theme 2: Formalization of the measurement process, Theme 3: Scope of the measurement process and Theme 4: Implementation support for formally capturing and analyzing knowledge.
3. There is an evolutionary pattern that measurement follows. We start with project, then product, and finally process measurement, and because of that the following themes are important: Theme 5: Measurement evolution within the organization and Theme 6: Measurement support for management control of software.

4. Project, process, and product improvement is achieved by using collected data as information that identify problem areas, and implementing mechanisms for the problem prevention based upon informed analysis of the product and process. Therefore the following themes are important: Theme 7: Project improvement using measurement technology, Theme 8: Product improvement using measurement technology, Theme 9: Process improvement using measurement technology and Theme 10: Predictability of project, product and process characteristics.
The maturity levels in this model are the same as the defined in the CMMI model. The peculiarity of this model is that for each theme was defined a stage in every maturity level (see table 3.2) and those stages may be followed by a software development organization in order to reach the highest level of maturity for that particular theme. Other characteristic of this model is that the maturity of the development process and the maturity of the measurement process are considered to be very closely related. The definition of the levels is as follow:
Level 1 - *Initial*: There exist no formal definition of the software development and measurement process and no process definition and measurement are conducted.
Level 2 - Repeatable: process definition and measurement are done at least at the project level.
Level 3 - *Defined*: Product measure is practice done in the organization using document standards like the Goal/Question/Metric paradigm.
Level 4 - *Managed*: process and metric practices are extensively used and managed. The process is measured and controlled and there is evidence of quality improvement.
Level 5 - *Optimized*: the organization have enough knowledge for the use of process and metrics and they are able to optimize the process and to use better measures.



Theme	Level 1 Initial	Level 2 Repeatable	Level 3 Defined
#1 Formalization of Process development	Process unpredictable No/poor process focus	Repeat previously mastered tasks Process depends on experienced people	Process characterized and reasonably understood
#2 Formalization of the measurement process	Little or no formalization	Formal procedures established Standards exist	Documented standards Standards applied
#7 Project improvement	No statistical process control No senior management involvement	Disciplined project and configuration management. Risk management	Dedicated process resources. Process data not retained nor analyzed properly

Figure 3.2: Example Themes and Level of Software Measurement Technology Maturity [MKDB90]

Assessment

For the measurement maturity assessment of an organization, this model proposes a list of questions ordered within each maturity level. Those questions make a relation between the maturity level and the corresponding theme or

themes, and they are formulated in a way that the answer must be either “Yes” or “No”. In order for an organization to be in the level i-th of the maturity model, at least the 80% of the questions corresponding to the level i-th muss be answer with ”Yes”. The following are examples of these questions:
“Is your organization able to repeat previously mastered task?” This question is related with theme 1: Formalization of the development process and with the level 2 (Repeatable) of the maturity model.
“Does the organization have dedicated software resources?” This question is related with the theme 7: Project improvement and the level 3 (Defined) if the maturity model.
After answering those questions, an organization should look at those that were answer with “No” and with the theme table identified which actions should be performed in order to evolve to a higher maturity model.

3.6 Discussion

An overview of the importance of SPI has been given. It is a very challenging activity for an organization and requires a lot of hard work, but it can be assisted by some models that guide in its implementation. Some models that help in the assessment and improvement of software processes (i.e. CMMI and SPICE) were also described. Based on these models, some other specific models for the assessment and improvement of the SPI strategy and the measurement process have been developed. They provide a basic for the organizations to assess the maturity of these processes and give a series of criteria that help to identify weaknesses in the process and to improve it. A critic related with the use of such a model, as mention in Lichter et al., is that those models are developed by a specific group of persons and reveals the experiences of them, but these experiences are not representative for the multiple existing software projects [HL07]. For example the model propose by Daskalantonaskis [MKDB90] was developed from experiences of the implementation of software processes in Motorola, but it is possible that another type of development software company has different characteristics and that this model not really fully applies. Nevertheless those models can be adapted in order to make them applicable to the measurement process of a specific organization, as Diaz et al. did with the Dakalantonaskis model [DLGP08]. If the models for the measurement process work for every organization, is another point that is still not clear. That is because in both of the models here presented, the maturity of the measurement process is closely connected to the maturity of the development process, and therefore if a company has a not very mature development process no really mature measures can be performed.
Important to remark is that a well managed and controlled process has direct impact in the quality of the final product, thus is necessary that the organization realized that and start initiatives in order to the improvement of those

processes.

Bibliography

[BR81] Christine V. Bullen and John F. Rockart. A primer on critical success factors. Technical report, 1981.

[Das94] Michael K. Daskalantonakis. Achieving higher sei levels. *IEEE Softw.*, 11(4):17–24, 1994.

[DLGP08] María Díaz-Ley, Félix García, and Mario Piattini. Mis-pyme software measurement maturity model-supporting the definition of software measurement programs. In *PROFES '08: Proceedings of the 9th international conference on Product-Focused Software Process Improvement*, pages 19–33, Berlin, Heidelberg, 2008. Springer-Verlag.

[GH01] Wolfhart B. Goethert and Will Hayes. Experiences in implementing measurement programs. Technical report, Technical Note CMU/SEI-2001-TN-026, 2001.

[HF97] Tracy Hall and Norman Fenton. Implementing effective software metrics programs. *IEEE Software*, 14:55–65, 1997.

[HL07] Lichter Horst and Jochen Ludewig. *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. Heidelberg, Neckar : dpunkt, 2007.

[MKDB90] Robert H. Yacobellis Michael K. Daskalantonakis and Victor R. Basili. A method for assessing software measurement technology. *Quality Engineering*, 3:27–40, 1990.

[NvV98] Frank Niessink and Hans van Vliet. Towards Mature IT Services. *Software Process – Improvement and Practice*, 4(2):55–71, June 1998.

[NWZ03] Mahmood Niazi, David Wilson, and Didar Zowghi. A model for the implementation of software process improvement: A pilot study. *Quality Software, International Conference on*, 0:196, 2003.

[NWZ05] Mahmood Niazi, David Wilson, and Didar Zowghi. A framework for assisting the design of effective software process improvement implementation strategies. *J. Syst. Softw.*, 78(2):204–222, 2005.

[PWCC95] Mark Paulk, Charles Weber, Bill Curtis, and Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, Boston, 1995.

Part II

Product Metrics

Chapter 4

Earned value in agile systems

Elena Soldatova

Contents

Abstract: Although the Earned Value Method is a well-recognized method of tracking the status of the engineering project, still agile software projects lack practice in using it. This article provides a review of the application of the earned value method as a metric for Agile development processes. Several techniques were reported to successfully have been applied. However there are suggestions for possible extensions.

4.1 Introduction

The usage of an agile approach for software development became a popular and inspiring technique followed by many companies in the last 10 years. Change of scope, unplanned budget and schedule are expected while performing such projects. Traditional project management methods recommend using a Gantt chart to manage the project schedule. Choosing or inventing appropriate metrics for measuring the progress for conditions of Agile development is a challenge. Studies exist around taking the existing metrics of economic theory, which are older and mature, and trying to adopt them. Earned Value Method is a well-known technique used for measuring the progress during a development process. However, other agile project management experts consider that old techniques could not be relevant to a new agile method and agile metrics are required instead.

The current paper reviews several methodologies of agile tracking based on EVM approach. This article studies the applicability of the EVM metrics for Agile projects. The traditional Earned Value method is based on assumptions of a fixed budget and a known scope. Studies done to use and adopt EVM formulas are investigated. The review of the articles providing thoughts about the limitations of EVM method is also given. As well the research of possible extensions suggested in literature is provided.

Sections 4.3 and 4.2 provide the background of the Agile development approach and the Earned Value method. Section 4.3 introduces several methods suggested for using EVM in Agile processes. Section 4.4 indicates the results reported by the authors of these methods. The discussion about limitations of the EVM and suggestions is provided in the same section.

4.2 Earned Value Method (EVM)

The EVM method was primary used by the U.S. federal government and Department of Defence (DOD) in engineering projects from 1967. Until now it's a part of the compulsory methods for governmental contracts. The EVM method is based on 32 criteria of the National Defense Industrial Association defined in ANSI/EIA 748.

The industrial standard was revised in 2005 in order to be compliant with up-to date projects. In [Wyn04] the threshold of 20M$ is estimated. It's used as a criterion for application of EVM. The application is considered to be a risk-based decision for projects lower the threshold and for the cases with schedule/cost overrun possibilities. Therefore a project-manager is a person taking his own responsibility on such projects.

4.2.1 Earned Value method concepts

Work schedule, scope and resources available often serve as leading indicators of whether a project is likely to achieve the financial goals. Earned value method provides metrics to reveal the performance upon these chosen indicators. For the earned value method the work is supposed to be split into the measurable delivery intervals or work packages. The measurements are performed with accordance to the schedule of work package delivery. Below

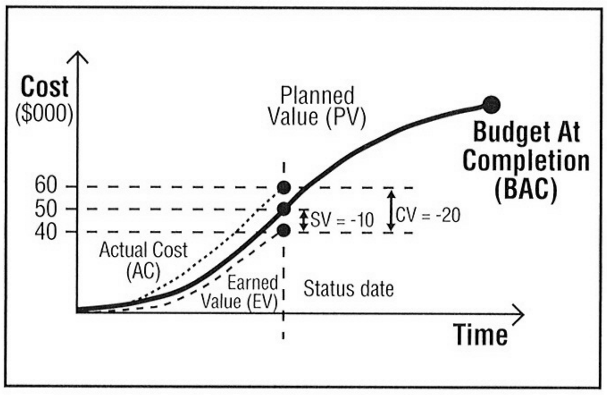


Figure 4.1: EV, PV, CV plot

the main terms that are helpful for understanding the EV method concepts are provided. The detailed explanation is given in [Anb04]. The first metric to be concerned is Planned value – the planned amount of budget assigned to be spent to fulfill the workpackage. When the Planned value is plotted over a time scale this results in the s-curve, which is also called a budget baseline or project management baseline (PMB). The second metric used is Actual Cost (AC), it represents the real amount of budget spent already. Earned Value (EV) represents the planned cost of real work which is performed at the current day. It can be described as EV $=$ (team efficiency) $\times$ Planned Value. Where "team efficiency" is used in remembrance to physical efficiency which shows how close the work really done to the the work planned.

The difference of Earned Value and Actual cost is called Cost Variance, the difference between EV and PV is Schedule Variance (SV). $CV = EV - AC$, $SV = EV - PV$. Estimate at Completion (EAC) is the total cost estimate based on actual cost and actual performance. According to figure (4.1) [Anb04], you can see that when Planned Value is 50, earned value is only 40, whereas actual cost is 60. Schedule variance is then equal to $40 - 50 = -10$, cost variance is $60 - 50 = 10$. The ratio of Earned Value to planned value (PV) is called Schedule Performance Index. The ratio of Earned Value to actual cost (AC) is called Cost Performance Index.

4.2.2 EVM benefits

Earned Value Management (EVM) is a well established and accurate measure of a project's cost performance. "EVM provides an early warning system for deviations from plan and quantifies technical and schedule problems in cost terms, providing a sound and objective basis for considering corrective actions (work arounds, tradeoffs and others). Therefore, EVM both mitigates the risk of cost and schedule overruns, while also providing a forecast of final cost and schedule outcomes."[Wyn04]

4.3 Agile Principles background

Agile development process is based on meetings with the customer and is split into iterations (sprints). Several types of agile processes exist, however they are united by common principles, therefore later Extreme programming (XP) and Scrum practices are concerned. The common principles of agile development processes are documented in [BBvB^{+}01]. One of the most important states: "Responding to change over following a plan". During the interaction between a team of developers and a customer user stories are worked out. User stories are then translated into the list of features, where each feature is assigned with story points characterising its difficulty. Story points are gained only upon delivering all the functionality associated to them. However, the agile nature of the process lies in a possibility to change or to expand a user story during the next meeting. Such alteration will probably result in the change of the amount of work scope to be done till the next review. A burn-up chart is a means used to plot functionality delivered, upon the time scale. The team velocity is determined by the curve incline. Agile development process focus on maximising return on investment (ROI) through continuous replanning and reviews, this approach allows to deliver highly-qualified software relatively fast.

4.3 Applying Metrics-based Approach to Agile processes

Metrics Based Scheduling and Management is one of the nine software practices introduced in the article [All99]. It's worth applying in order "to identify problems early in the development lifecycle." The drawbacks of XP and Scrum methods are presented in a detailed study provided in the article [VC04]. One of the most important being pointed out is the lack of a feedback after each sprint. Therefore it's highly desirable to be able to determine how well an agile method is working; in general it would be valuable to determine whether iterations of agile processes need improvement. The agility of the development requires that the metrics should be visible and understandable for developers, managers, and customers. During agile process there is no time allocated for teaching the team with difficult concepts. The application of the metrics should easy as well.

4.3.1 Premises for connecting Agile processes and Earned Value

Due to the benefits of earned value method it's desirable to find out if the means to use it in agile processes exist. A prerequisite for applying EVM is defining scope up front and having a stable baseline plan. On contrary, agile plans are rarely stable and often lack a fined fixed and clear scope. As well they are usually implemented within small companies what is treated as a risk for the EVM standard application. That's why applying EVM metrics requires attention to these problems and preliminary study. However, agile processes and earned value method appear to have more things in common than it seems at the first glance.

Several authors pointed out the premises that allow using Earned value in Agile projects. For instance, it was pointed out that a Burn down chart of Agile development process is equivalent to a EV graph. The essential difference is that people traditionally include on the earned-value chart "tasks completed", whether or not they resulted in code getting integrated. On agile projects, credit is only given when code is integrated and tested (the "no left-over sock" rule)[Coc04]. In [All93] the author identifies how to connect Agile process with Earned Value. At the end of each sprint a unit of work produced is self-contained and verifiable upon requirements, therefore it's taken as the main measure for Earned Value. Main issues of Agile processes needed to be taken care of are: a proper iteration length, choice of the cost of each iteration effort. The recommendations above lead to the in the next section.

4.3.2 Simplified EVM for Agile Development

[Rus09] reports about a simplified way of using EVM metrics in Agile projects. He keeps track of project by using a traditional Agile burn chart enriched with two more lines. When introducing method to users, the author doesn't complicate the explanation with EVM terms. The cost of the work done during the week is calculated based on email reports from team members about work hours. In order to build the black line (fig 4.2) the breakdown structure is not used. The earning rule for the team is that points are only given for working software. The estimation of future costs is done by linear extrapolation.

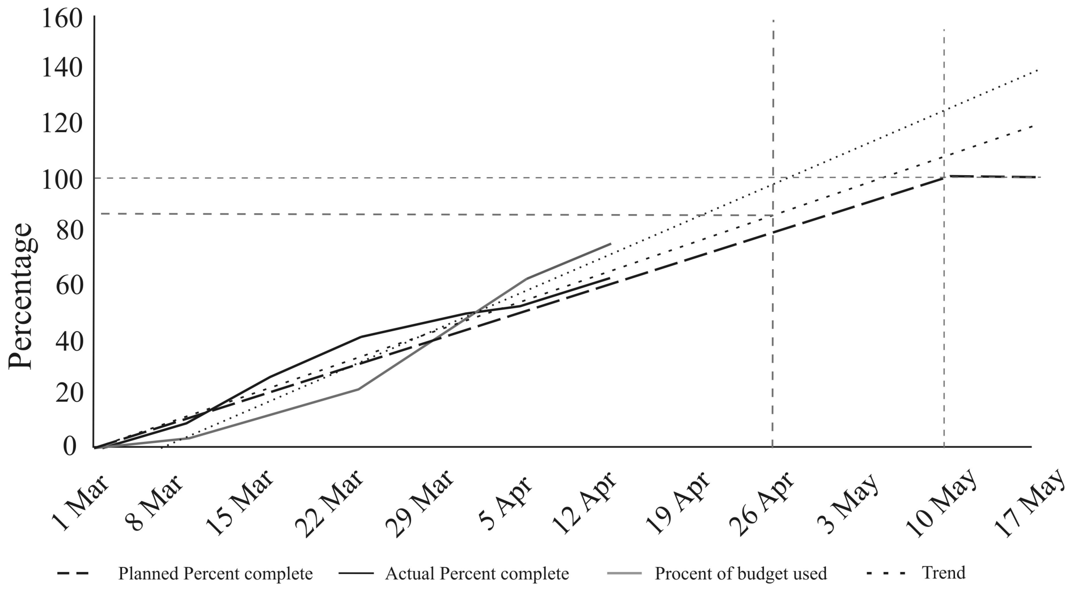


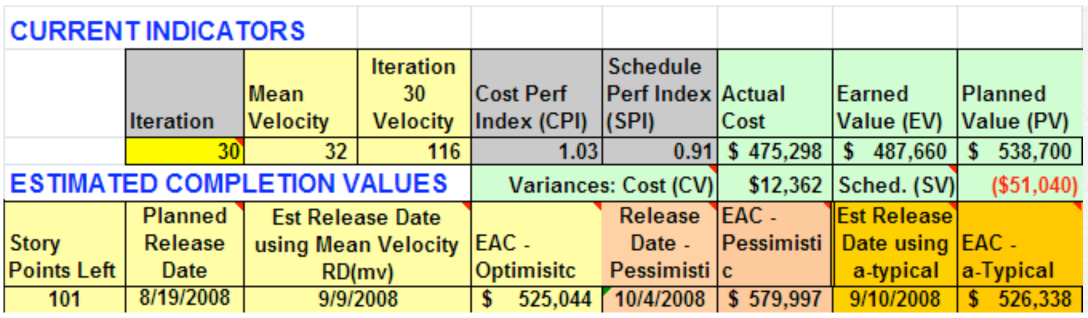
Figure 4.2: Burn chart

1) The dash line shows the progress that is expected, as a percentage of the total work progress. It starts at zero and slopes up to 100% at the end of the project. 2) The grey line shows how much of the product has been built in percents of the total product size. Ideally, it should follow the same path as the dash line — starting at zero and reaching 100% on the project's scheduled end date. If it falls below the grey line that means that the team is below the schedule and visa versa. 3) The black line shows the cost incurred as a percentage of the total project budget. Ideally, it would also follow the same path as the dash line — starting at zero and finishing at 100% of budget, on the project's scheduled completion date. If the black line is above the grey line that means the budget is spent faster than the team is building the software.

In the example (fig. 4.2) it can be seen that the project will probably complete earlier. It can be estimated visually that in the end only 85% of budget will be used.

4.3.3 Agile EVM in Scrum and XP projects

The article [SBB06] appeared in 2006 and so far is the most complete analyzation the question of applicability pure EVM method formulas to Agile process realities. The correlation between AgileEVM and Scrum formulas for the release date estimate is proved in it. This proof mathematically allows to use Standard EVM definitions and Formulas in Agile Projects. The author provides a worksheet in order to ease the calculation and tracking of the results. The following parameters are used for initialization: Budget at Complete, iteration length, number of planned iterations, start date and planned story points to be achieved. At the completion of each Sprint, four data points are entered: sprint number, points completed, points added, sprint cost. The worksheet automatically calculates baseline schedule. Information provided is given in the picture 4.3. In the section ”current indicators”, the available characteristics for this sprint are provided: iteration, mean velocity, velocity of current iteration. It can be seen that the cost performance index is 1.03 which means that there is budget underrun, while schedule performance index is very close to 1, it is 0.91. In the section ”estimated completed values” the forecasts are done based on current indicators and AgileEVM formulas. The estimated parameters include number of points left, planned release date. The worksheet as well includes the results obtained by using different formulas to compute the Estimate at Completion. The problem of calculating EAC is described in the page 64.



CURRENT INDICATORS								
	Iteration	Mean Velocity	Iteration 30 Velocity	Cost Perf Index (CPI)	Schedule Perf Index (SPI)	Actual Cost	Earned Value (EV)	Planned Value (PV)
	30	32	116	1.03	0.91	$ 475,298	$ 487,660	$ 538,700
ESTIMATED COMPLETION VALUES				Variances: Cost (CV)		$12,362	Sched. (SV)	($51,040)
Story Points Left	Planned Release Date	Est Release Date using Mean Velocity RD(mv)		EAC - Optimisitc	Release Date - Pessimisti	EAC - Pessimistic	Est Release Date using a-typical	EAC - a-Typical
101	8/19/2008	9/9/2008		$ 525,044	10/4/2008	$ 579,997	9/10/2008	$ 526,338

Figure 4.3: AgileEVM worksheet

4.4 Applicability discussion

In [Dym08], [HD06] the author claims that EVM estimates are quite inaccurate due to constant changes in the customer requirements. The author points out that it’s possible to calculate Actual market acceptance and actual ROI very quickly through implementing most highly valued features firstly. The author is slightly incorrect because in his mind EVM is only keeping to Schedule and Cost Performance. Therefore he insists on dropping out EVM and focusing on Value delivered and on Usage of Velocity as diagnostics to forecast how much work they expect. As well it can be seem that the author is contradicting himself, naming the same EVM principles that already are used. Moreover, a successful usage of EVM metrics together with Earned Business value is demonstrated in [Raw08](subsection 4.4.3).

Fleming identified main criteria for implementation of the EVM metrics in Agile processes [FK98].

1. continuous management of the work remained i.e. project manager should have enough expertise to take measures in time (an example is provided on the page 64)
2. qualitative project's baseline plan which is maintained due track of changes (an example provided on the page 64)
3. periodical measurement of actual performance against the baseline plan (see example in the subsection 4.4.3)

Regarding the first criteria, in the article [Cuk09] the author attracts attention to the importance of Integrated Baseline Reviews as a better way of getting understanding of a Performance Measurement Baseline (PMB). Term "Integrated" relates to the team: the review should include members from the technical, financial, scheduling, risk management departments, and contracting communities.

4.4.1 Simplified EVM discussion

The aim of Rusk in [Rus09] was to gain insight in the financial aspect of the work on project. His method allows to understand whether the project is on track, whether the progress done by the team is enough. As well it allows to distinguish between different reasons of the underrun.

The author avoids using EVM acronyms to keep the approach simple. Percentage-based approach is easier to understand than approach suggested by [SBB06]. However the technique allows to involve all the team, stakeholders and managers in the discussion process as the useful metrics such as EAC, TCPI are derived visually. Burn-up chart and costs, together with burn-up chart and schedule variances were found to be enough to realize that changes are needed.

The applicability of the technique depends on the possibility to define the scope up front and on a linear delivery of value. The approach "relates cost, schedule, technical accomplishment in the same way as classical EVM — albeit with a linear schedule rather than the classical s-curve." The technique is as well cost-effective for projects with budget smaller than $20 million. According to the standard ANSI/EIA748 due to its costs the standard EVM can be omitted. The chart works best when it covers the period of 3-6 months.

Article [Sol09] provides discussion for [Rus09]. P. Solomon points out several weak points in the article of John Rusk: the assumption of linear delivery of value and the habit of implementing unforeseen requirements as they arise. Solomon focuses on accurate reporting of project status. For software based on incremental builds, the author recommends allocating resources and defining baselines for each of the builds. Functional requirements must be also allocated and prioritized within each of the builds. Before starting the build the consistence with the total project schedule should be verified. These recommendations allow to use EVM critera for project with non linear PMB. If functionality is deferred from one build to another, then the budget is transferred as well to another build. When the rework for the earlier accepted story points is required, then Negative Earned Value is introduced. This allows to maintain aggregated Baseline.

4.4.2 AgileEVM discussion

The contributions of AgileEVM method compared to Scrum Framework itself are the following:

1. allows to use the data achieved (Cost Performance Index and Schedule Performance Index) to validate the project status
2. allows to forecast the budget spends using Estimate at Complete and Estimate to Complete
3. allows to be more versatile in identifying future strategies for developing the product

The combination of CPI and SPI with a burn-down chart provides additional information enabling the project manager with information to take measures for reducing scope or to adjust a planned release date.

This method is particularly useful for budget responsible persons such as project manager. The agile EVM provides the team of developers with metrics that allow quickly to respond to the loss of velocity in development. Additionally it doesn't add burden to usual developers as it's easy to be reported.

In the project where AgileEVM was used, the following result was achieved: "Adapting the traditional earned value management metrics to the Scrum project allowed the ScrumMaster to accurately predict the impact of these requirement changes, despite a fluctuating velocity, on budget and schedule for the release. The ScrumMaster was able to then share this information with the

Product Owner and the team, giving them information to help make additional decisions."([SBB06])

In [SSS^{+}07] Sulaiman et. al. points out AgileEVM is not necessarily appropriate for all Agile projects. Projects with extremely short release cycles will not find as much utility from this technique as projects with longer release cycles involving multiple sprints or iterations. In my opinion that though the method Tamara Sulaiman presented is provided in all the details needed, her further articles presenting the results of real projects are very concise in reporting details and values.

4.4.3 Further implementations and metrics

Earned Value Management metrics should be used in conjunction with Earned Business Value. In [Raw08] author goes further the studies done in [SBB06] and suggests new methods for calculating CPI and SPI assuming consistency across the sprints. $CPI = \frac{BC}{SP} / \frac{AC}{SP}$ instead of $CPI = EV/AC$, where BC/SP – is Baseline Cost per Story Point, AC/SP — actual cost per Story Point. CPI indicates if customer are getting the SPs they are paying for. $SPI = \frac{AV}{BV}$ instead of $SPI = EV/PV$, measures if developers are getting the Story Points at the rate expected, where AV is Actual Velocity and BV is Baseline Velocity.

The article is valuable for adding Earned Business Value to measure efficiency of delivery. EBV is defined as "the percentage of the product's Business Value that is currently earned". The BV metric was introduced in [Raw06], where BV is assigned only to those of WBS stories that deal with implementing features or with marketing activities. According to S-curve and Pareto Rule Rawstorn

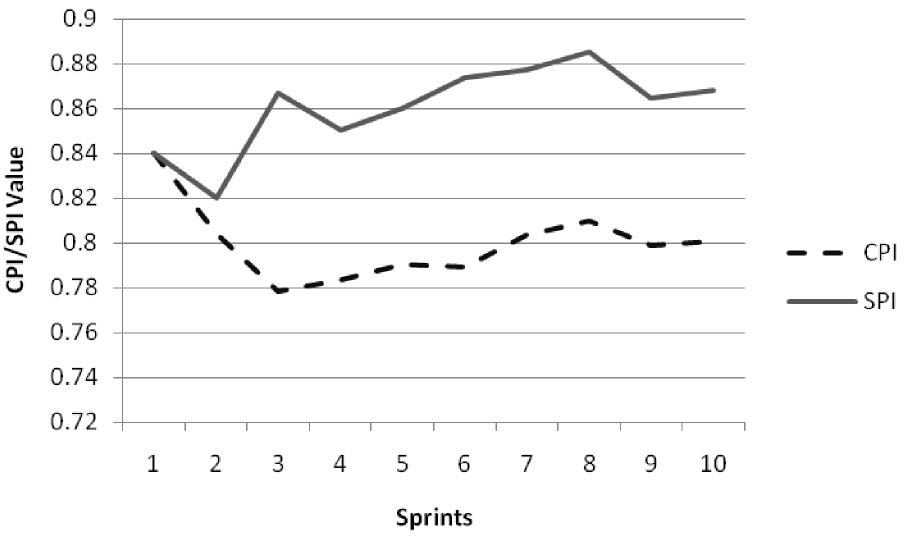


Figure 4.4: CPI and SPI graphs for non-ideal release

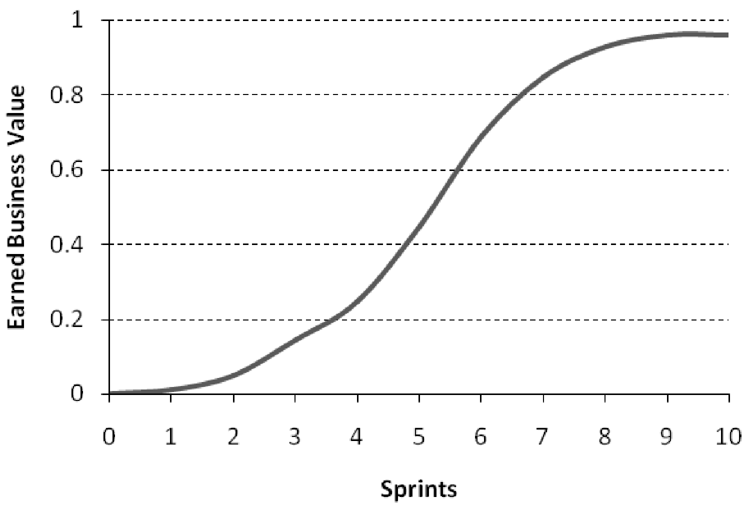


Figure 4.5: EBV graph for non-ideal release

provides example of non-ideal situation, where fulfilling story points in order to maximise BV leads to better fulfillment though with latency in CPI and SPI (fig.4.4, fig. 4.5.)

In work [Car] is reported about development of a tool support for EVM in Agile Processes within xProcess toolset. Distinguishing between Planned Cost and Planned Value is suggested. Planned Cost is the budgeted expenditure, and Planned Value is the proportion of Planned Cost corresponding to the size of the task done. The idea is similar to idea of [Raw08], to distinguish between work done and overhead introduced. Therefore formulas for CPI and SPI were modified as following: $CPI = PlannedCost/ActualCost$, $SPI = EarnedValue/PlannedCost$. The environment provides plots to compare planned values against actual, to forecast the Earned Value and Actual cost Curves.

4.4.4 Extensions for using EVM in Agile Processes

It was pointed out on the page 60 that quality of project's baseline is of importance for implementation of the EVM. Therefore it's meaningful to investigate how to apply EVM when a budget baseline is uncertain. In addition, it's worth paying attention on choosing among the ways to estimate the final costs (EAC) and performance. This can give additional evidence of current actions needed to be taken. Below several methods that represent efforts of improving the usage of EVM metrics in Agile processes are presented.

Baseline and Quantity Adjusted Budget

The suggestions for projects with a highly uncertain budget baseline are given in [Kan07]. Quantity Adjusted Budget allows to modify the baseline by evaluating how it was changed since the last release.

Firstly, each of the features identified in work packages, is assigned with some quantity to be done. After each iteration the adjusted number for each feature is acquired, and the budget is recalculated. The new budget is obtained by multiplying the new estimated quantity with the budget unit rate: $QAB =$ (Adjusted quantity) $\times$ (Budgeted Unit Rate). Where for quantity stand not only program features as well amounts of hardware materials (servers, hard drives), software licensing (database engine, reporting tools, society application). The number of features to be realized determines the amount of work hours required by the developers' crew. Earned Value $=$ the percentage of completion$\times$ QAB. The calculations for SV,CV, CPI, SPI remain the same as for fixed budgets, however QAB allows to update the budget smoothly after each iteration done.

Ways for Estimating Estimate at Complete

Numerical accuracy of the EVM formulae used for estimating Estimate at Complete value is examined in the article [Chr93b] and [Chr93a]. The author gives the details of a project, which failed because of managerial faults in application of EVM. The Navy's program manager responsible for an advanced aircraft project chose to rely on a lower estimate at complete, despite several higher ones prepared by his own analyst. This led to huge overrun of a budget and to cancellation of the project before its completion.

The author reviews three closed reports that were evaluating accuracy of the various formulas calculating EAC on real-life projects. The need to calculate the EAC is based on the problem how to use the current actual costs, earned value and budget at completion to identify the actual costs in the end of the project. Usually the actual costs differ from planned, therefore several adjustment indices can be used.

The to complete performance index (TCPI) is "the efficiency required to achieve the EAC planned". The TCPI is useful for comparison with CPI to evaluate whether EAC is reasonable or too low. The author introduces a simple rule, saying that when the difference of TCPI and CPI exceeds ten percent this is a sign to reconsider EAC.

The research gives data on how "accuracy of index-based formulas depends on the type of the system, the stage and the phase of the contract". Author found out as that accuracy of regressive methods compared to index-based

EAC formulas is a matter to research. In general, while choosing a EAC derived from different formulas author warns from impairing a culture that suppresses truth, because a project manager should be able to interpret about the signs of overrun.

This article provides interesting results for consideration however it was issued in 1993 and the problems indicated in it still need to be investigated.

4.4.5 Future reading

Main people, communities participating in Agile EVM:
1. SolutionsIQ, organisation providing training on Scrum projects, provides a library of white papers, articles, and presentations. Actively promotes usage of AgileEVM. Tamara Sulaiman, Thomas Blackburn work there.
2. The Data and Analysis Center for Software (DACS) – allows free register and download of the journals.
3. There is a nice resource which gathers all the literature concerning earned value analysis. http://www.suu.edu/faculty/ChristensenD/EV-bib.html It was beyond the scope of this article though the interested reader is addressed to it.

4.5 Conclusion

This article provides a review in the field of application of the earned value metrics in agile development processes. Earned Value Formulas have been demonstrated to be appropriate to Agile process and useful for the stakeholders of the project. However, it was noticed that a lot of authors focus on CPI and SPI Earned Value metrics for evaluating performance and in order to ease the application. It was shown as well that an additional metric such as Earned Business Value responds to real agility of the project. Several extensions of the traditional Earned Value metrics together with their finer points are provided. Identifying more clear criteria for choosing proper iteration length, formulas for EAC is still work needed to be done. Moreover, there is a significant lack of tools implementing EVM metrics, this fact was also noticed by [CG06]. For the current moment there is only a number of self-made worksheets, for example for AgileEVM, and an experimental plugin to xProcess framework.

Bibliography

[All93] Glen B. Alleman. Project management == herding cats. http://www.pmforum.org/viewpoints/2003/0203agilepm.htm, February 1993.

[All99] Glen B. Alleman. Nine best practices. The Software Management Framework, June 1999.

[Anb04] F.T. Anbari. Earned value project management method and extensions. *IEEE Engineering Management Review*, 32(3):97–97, 2004.

[BBvB$^+$01] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, and Others. Manifesto for agile software development. *The Agile Alliance*, pages 1–2, 2001.

[Car] A. Carmichael. Evm and agile processes - an investigation of applicability and benefits.

[CG06] A. Cabri and M. Griffiths. Earned value and agile reporting. In *Proc. of AGILE Conf*, volume 06, page 6. Citeseer, 2006.

[Chr93a] David Christensen. Determining an accurate estimate at completion. *National Contract Management Journal, Arlington MA*, pages 25:17–25, 1993.

[Chr93b] David Christensen. The estimate at completion problem: A review of three studies. *Project Management Journal, Project Management Institute*, pages 24:37–42, March 1993.

[Coc04] Alistair Cockburn. Earned-value and burn charts. http://alistair.cockburn.us/Earned-value+and+burn+charts, June 2004.

[Cuk09] Anita Cukr. Dispelling Some Myths about Earned Value Management (EVM). *SoftwareTech*, 12(3):5–7, 2009.

[Dym08] Robin Dymond. Earned Value Measurement — the useless metric for Agile, 2008.

[FK98] Quentin Fleming and Joel Koppelman. Earned value project management a powerful tool for software projects. *Crosstalk – The Journal of Defense Software Engineering, Ogden, UT, http://www.stsc.hill.af.mil/crosstalk/1998/07/value.asp*, July 1998.

[HD06] D Hartmann and R Dymond. Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. *In Proceedings of the Conference on AGILE 2006 (July)*, 23(March 1993):28, 2006.

[Kan07] B. Kane. Estimating and Tracking Agile Projects. *Citeseer*, IX(V):1–15, 2007.

[Raw06] D. Rawsthorne. Calculating Earned Business Value For An Agile Project. *Agile Journal, June*, pages 1–6, 2006.

[Raw08] D. Rawsthorne. Calculating Earned Business Value For An Agile Project. *Agile Journal, June*, pages 1–5, 2008.

[Rus09] John Rusk. Earned Value for Agile Development. *DACS*, 2009.

[SBB06] Tamara Sulaiman, Brent Barton, and Thomas Blackburn. Agileevm — earned value management in scrum projects. In *Proceedings of AGILE 2006, AGILE '06*, Apr. 2006.

[Sol09] Paul J. Solomon. Agile Earned Value and the Technical Baseline. *SoftwareTech*, 12(3):9–12, 2009.

[SSS$^+$07] Tamara Sulaiman, Scrum Schulungen, Hubert Smits, Product Owner, Scrum Framework, Text Issues, and Rally Software. Measuring Intergrated Progress on Agile Software Development Projects. *Methods & Tools*, pages 1–4, 2007.

[VC04] M. Visconti and C.R. Cook. An ideal process model for agile methods. *Lecture notes in computer science*, 3009:431 – 441, 2004.

[Wyn04] Michael W. Wynne. Proposed revision to dod earned value management policy and rationale for changes. *Defense AT & L*, pages 1–5, 2004.

Chapter 5

Architecture

Ricardo Tavizon

Contents

Abstract: Software architecture refers to a form of abstraction describing the overall components from which a system is built, the relationship among those, the patterns guiding their composition and constraints to the non-functional requirements. It is necessary to conduct an evaluation in order to know if an architecture is suitable for a system. An architecture evaluation should be a standard part of the architecture-based software development life cycle according to [CKK01].
This paper is intended as a contribution to review some types of architecture evaluations dedicating some space to static evaluations and mechanisms to conduct them.

5.1 Introduction

Software architecture is one of the most relevant assets being developed which is actively used during life-cycle phases, with strong influence in the implementation phase. Architecture is seen as a mean of specifying the system, understanding it and providing communication from the high level aspect to the stakeholders. For the same reason an evaluation should not be seen as a trivial task, as in many cases is not performed. However, not to carry out an evaluation might lead to problems which in later phases (i.e. development) would imply more costs, time and resources. Decisions taken on the architecture have considerable impact on the quality attributes associated to it. Evaluation tries to establish mechanisms that allow to study the impact of the decisions based on the expected results.

Software metrics have been applied more often in the implementation level and those measuring techniques and models over the past decade have been pushed up to evaluate architecture itself. On this field quantitative over qualitative metrics result to be more mature. It is important to notice that this measuring techniques require more effort. Among a broad range of evaluations, there is one denominated static. Static evaluation compares two models, the planned architecture against the implemented one. This is done by the meaning of a mapping between the two models giving as a result a report stating whether the comparison converges or diverges. For this comparison the use of human-based task and tools is required.

This paper starts in section 2 by providing an overview on the term architecture. That is to support the need for evaluation, which is described in the following sections. Also this section includes the different techniques for evaluating architecture and the selection criteria that should be followed. The section is closed explaining the overall process on any of the evaluation techniques. Section 3 discusses evaluations of the type static, scenario-based and architectural assessment techniques. Section 4 illustrates static evaluation by the use of tools. Section 5 provides some space for discussion, while section 6 discusses related work and concludes this experience report.

5.2 Background

First, the term of software architecture is provided. Although there is no widely accepted definition by the industry, two definitions are examined: one from the professional body IEEE and the other proposed by Bass et al. respectively. In the author's opinion, both definitions allow to understand architecture as the blueprint for the system to be developed and its ongoing project.

"Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" [IEE00]

This definition is intended to present what constitutes the foundations of a system: the components (which either are physical or logical), their way to communicate and the organizational structure they follow.

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [BCK03]

Architecture according to the previous definition is the core of the system to be implemented. After the requirements specification has been completed, the next step leads to designing the architecture. Three benefits are gained from the design. First, the architecture will be used as a basis for communication between the stakeholders. That is, it stands for the common model being planned. Second, all the decisions related to design will be done early. It is known, that all the decisions concerning the design will have a great impact in later phases (development, deployment and maintenance). Finally, architecture can be seen as a model. This model can be applied to other systems conforming similar requirements.

It is important to notice that there is no term who makes mention of whether an architecture is right or wrong and if it will enable the system to fit the expected requirements or not. The systems being delivered might deal for instance with lack of performance or security vulnerabilities as a result of architecture decision. The best way to choose an architecture is done by evaluating it. An evaluation is a way to assure that the architecture meets its functional and non-functional requirements.
Next, the benefits that can be gained when evaluating an architecture are exposed. The entry point is the validation of requirements. Requirements are presented as a list and some clarification and prioritization will be done solving any conflict that it might exist among them. Reduction in costs is associated with time and committed resources.

The earlier a potential issue in the candidate architecture is identified, the better to validate the feasibility and saving in resources. It is all about risk mitigation. Preparation is required prior to any formal evaluation, this clearly

increases the understanding of the system and leads to better documentation which allows to see the design choices and their justification. During this evaluation, which it focuses on specific areas, it is possible to have a better perspective of the system's architecture and any required modification can be assessed following the rationale behind the structure. In other words, evaluation forces to manage a top-level architecture document that is understandable clear. Detection of potential issues in the quality attributes during an evaluation include extensibility, portability and security among others. The solution to any of those issues during late phases of life-cycle would imply more effort. The avoidance of these issues clearly allows to provide the capabilities and limitations from the system.
For further reading on different points of view regarding the term architecture it is worth to look at Software Engineering Institute [SEI].

5.2.1 Evaluation techniques

This section presents an overview of the available evaluation techniques and the importance of each of those for architecture (see Figure 5.1): two basic categories surround this frame used to perform an evaluation. Qualitative techniques which ask some questions to derive an elaborate report of an architecture and quantitative techniques applying some measurements to the architecture.

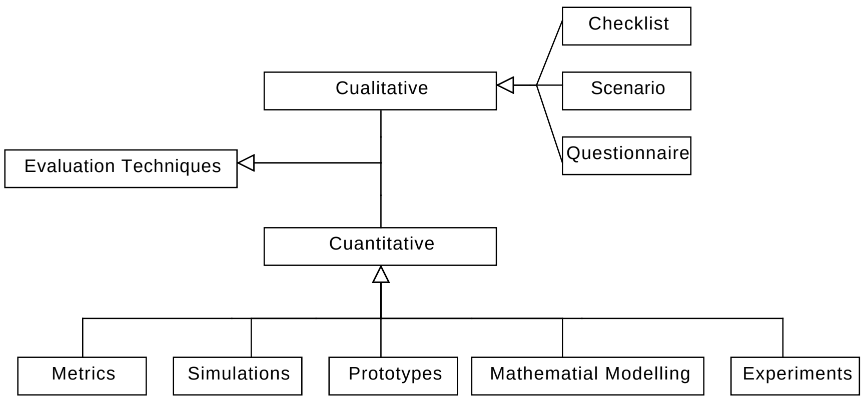


Figure 5.1: Architectural design method

Qualitative is based in questioning to assess the architecture for a given quality point and it can be performed as scenario, questionnaire or checklist. Those techniques differ on applicability to understand whether the architecture complies to its requirements. The scenario technique is specific to the system being evaluated while the checklist and questionnaire can be derived as domain specific entries. Questionnaires or checklists are supposed to exist before the project starts reflecting a more evolved evaluation practices, while scenario-based is developed as part of the system.

Scenario-based is used to try to assess specific attributes such as performance, security, modifiability and so on.

Questionnaire focuses on general purpose issues, like the way the architecture was generated and documented. An example from this technique is the software risk evaluation process, developed by the Software Engineering Institute [Abo97].

Checklist requires more preparation and understanding of the domain area. This technique tends to be more specific when assessing qualities from the architecture.

Quantitative follows the opposite direction compared to qualitative. Rather than provide ways to generate the questions that will be asked during an evaluation about an architecture, this technique provides answers to questions about particular qualities of an architecture. The classification includes metrics, mathematical modeling, simulations, prototypes and experiments.

Software metrics are quantitative methods that have proved to work in domains such schedule, size and complexity, cost and quality. During architecture evaluation common measurements include cyclomatic complexity, cohesion and coupling.
Simulation techniques could be used to assess performance and a prototype might be the model. Simulation and prototype techniques are used to compliment answers during a review or as evidence to support the architecture.

Now, evaluation can be performed during three different stages: early, middle and post-deployment. The stages reflect a iterative development process.

In the early stage, the only aspects that can be evaluated are architectural decisions and preliminary choices. During the middle stage some architectural design has been materialized as part of an iterative process and the evaluation is done at any time after the first iteration. The last stage, post-deployment, occurs once that the system has been designed, implemented and deployed. At this point, the model and the system are available and by meaning of mapping and comparison can be deducted whether the implemented architecture matches the planned architecture.

Which evaluation technique should be applied cannot be answered with precision. However, there are some recommendations: if some prototype, experiment or simulation was done during the development process, then this could be shown during an evaluation as part of evidence and to address specific questions. For example if performance simulations were executed during development process.
For a company that has no high experience performing evaluation practices,

the point to start might be to build some scenarios. Having a collection of scenarios would increase the expertise on the field and later on some transformation should be done from scenario-based techniques to questionnaires and checklists.

An example on using metrics during the architecture evaluation would be while the developers are implementing the system. In this scenario, the developers have immediate comparison between the source code model and the high level model (architectural design). Violations to the high level model would be identified during implementation. Necessary corrections would be applied during early stages. This evaluation is rule-based and it is explained in static evaluations section.

5.2.2 Evaluation process

The process to be followed is decomposed of mainly three parts according to [SF94]:

Analysis of all elements that will permit the understanding of the evaluation context. There are two types of evaluations: planned and unplanned. The first type is part of the project plan and the budget, this tends to be preventive and constructive at the same time. The unplanned evaluation can be seen as the existence of some trigger to perform it, let's say the usage of resources deviates from the plan. The participants of such evaluation are one element to consider. Representatives integrated by the architect, designers from the major components, stakeholders and the evaluation team. It has been suggested by [Abo97], that the evaluation team should be external to the organization. The purpose of the evaluation is another element and it needs to be clearly specified including all the goals and requirements so the scope of the same can be kept under control. As precondition, the preparation of material needs to be done.

Execution is the second part. This part refers to the execution of the evaluation and follow up of the activities by keeping tracking of all the issues and requirements being reviewed. It is recommended, as a good practice, to set some priority on each requirement. During the presence of a serious issue concerning the requirements, a valuation of modifying either the requirement or architecture takes place. It is important to notice, the modification of a requirement would be done only when it is possible and usually are fixed and cannot be negotiated. This leads to modification of the architecture to fit the requirements. When the architecture needs to be changed, this would mean that conflicting requirements are being solved.

Results are the last part of the evaluation process. The results include a formal report, enhanced documentation, preliminary system predictions and ranked issues.

5.3 Architecture Evaluation

The aim of performing evaluation at architectural level is to provide robust systems conforming not only functional requirements but also those requirements of the type non-functional. An evaluation is able to determine how adequate is the architecture in question for its intended purpose. This allows better quality, maintainability and scalability among other benefits.

Every system has an architecture, but not in all the times the architecture is defined and explicitly thought about. Part of the architecture delivery process should include testing it as early as possible to circumvent potential issues that might occur. For example, usage of tools to get useful quality metrics to tune up the code when necessary and provide changes. The idea for optimizing would be to take some metrics before and after changing something to see that the optimization made a significant effect. Taking metrics are necessary before any change is applied. Otherwise, the optimization cannot be measured. This section explains some of the evaluation techniques. Scenario-based is introduced and all the types of measuring techniques are covered.

5.3.1 Architectural assessment techniques

The transformation from requirements to architectural design seems to be less formalized than the other phases of rational engineering which follow methodological and technical support. This method deals with requirements which are directly influenced by the architecture of the system. On this case, the architecture evaluation is performed by means of simulation, reasoning, scenarios and mathematical modelling. The evaluation is applied over the functional and non-functional requirements (NFR) by improving the architecture using certain transformations in its design until the quality attributes or system properties are fulfilled. Functional requirements define a function from the software system or one of its components, so it is application oriented. NFR specify the criteria that can be used to judge the qualities of the system that are relevant for the development (i.e. maintainability, reusability, etc.) and the operation of the system (i.e. performance, fault-tolerance, etc.). Since the NFR cannot be matched to some part of the software system, the relation with the functional requirements is to impose constraints in the design or implementation. All the software systems need to comply with multiple NFR. Available design methods [JCJv92] only shed some light on achieving the system functionality without considering NFR. Due to this lack of support on the design methods, there exists a great risk of systems not compelling with the specified properties.

The goal of the method proposed by Bosch and Molin is to iteratively assesses whether an architecture supports each of the NFR and improves the architecture using transformations until all the NFR are fulfilled. The idea of

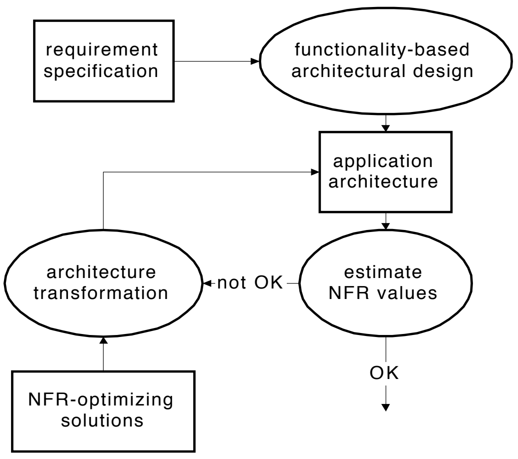

requirement specification
functionality-based architectural design
application architecture
architecture transformation
not OK
estimate NFR values
OK
NFR-optimizing solutions

Figure 5.2: Architectural design method

the method can be seen in Figure 5.2 [BM99]. The process is based on certain inputs and outputs and works as follows: the requirements specification is the first input to be considered and as a result the architectural model (known as functionality-based architectural design) is being output, which it will be implemented later on to have the application architecture. At this level, the design is evaluated against specified NFR. For each NFR, an estimate is given which can be compared to the values in the requirements specification. If all the obtained estimations are greater or equal than those from the specification, then the process is finished. Otherwise some transformation needs to be performed on the application architecture and the comparison is effectuated again until the estimations comply. As it can be seen, this is performed iteratively. When many iterations are done without complying the estimations, the evaluation could give as a result an inform stating that no feasible solution can be obtained.

Now, an estimation on the NFR has been mentioned and some measuring techniques are used for assessing the NFR and the approaches are: scenarios, simulation, reasoning and mathematical modelling.

Scenario-based technique develops a set of scenarios to materialize the meaning of the NFR and needs two different sets: one for the design and another version for the evaluation. The estimation of the NFR can be based on the radio between the succeed and failed scenarios. This evaluation is frequently used for properties such as maintainability and flexibility. This technique is explained in detail in the next section since it is used in the Architecture Trade-off Analysis Method.

Simulation technique divides the component into two groups. The first group is conformed with the main components, which must be implemented. The second group is indeed simulated. The result is an executable system which can be used in a simulated context to evaluate the application behavior under different circumstances. It is important to notice, that the simulation defines a precise interaction and behavior from the architecture and it is frequently used to evaluate performance and fault-tolerance.

Mathematical modelling is seen as an alternative to simulation and allows static evaluation of architecture. When a mathematical model is given, the evaluation consists to describe the system accurately.

Reasoning , also known as objetive reasoning, is the last from the list and goes out of the scope from this report. It is based on logical arguments from experienced software engineers.

To summarize, once that the system has been implemented, some testing is performed in order to determine whether the NFR are fulfilled. When this goal is not achieved, components from the system need to be redesigned in an iterative process.

5.3.2 Scenario-based: ATAM

Architecture Trade-off Analysis Method (ATAM) was developed by the Software Engineering Institute [KKC00] and is a scenario-based method for assessing NFR such modifiability, portability, extensibility and so on. The purpose is to involve the stakeholders in order to acquire the architecture by analyzing the business goals (functional and non-functional requirements) and the list of quality attributes used to generate scenarios.

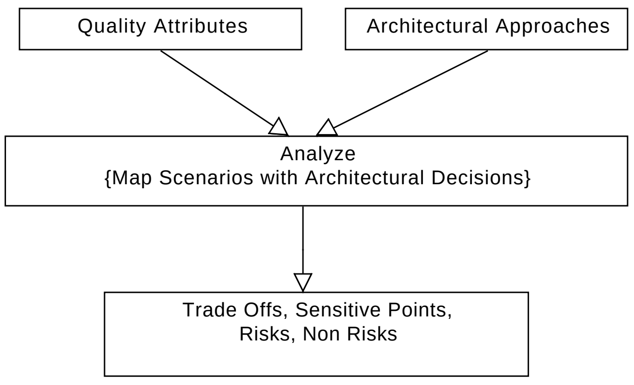

Quality Attributes
Architectural Approaches
Analyze
{Map Scenarios with Architectural Decisions}
Trade Offs, Sensitive Points,
Risks, Non Risks

Figure 5.3: Architecture Trade-off Analysis Method

The process is illustrated in Figure 5.3. It takes as input the business goals and different architecture designs. The business goals are specified in form of scenarios and prioritized. Analysis of each scenario is performed by the evaluation team, rating them by priority. The architecture is then evaluated and potential risks, sensitivity points and trade-off points are identified. Some iterations starting from the analysis step are performed. As a result, the evaluation team summarizes and presents the findings to the stakeholders in the form of scenarios based on the NFR and documented architectural designs. But how to solve the problems is out of the scope from the method.

The benefits from the method are that the stakeholders have more understanding on the architecture, the documentation from the architecture is improved, communication between stakeholders is increased and mitigates risks in early phase during life-cycle.

5.3.3 Static evaluations

The idea, similar to the architectural design method (Figure 5.2), is to compare the planned architecture model with the implemented source code model by mapping both representations.
The models are either postulated from the architectural design or obtained from the source code and both are composed of elements and relations in between. For each relation and model element, there will be a classification

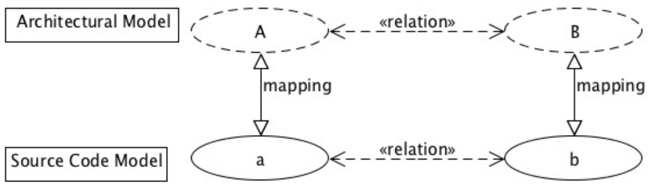


Figure 5.4: Static evaluation mapping

based on three types: the convergence, divergence and absence (see Figure 5.4). The convergence means the existence of the relation between the pair of components (A,B) and (a,b) from both models. Divergence specifies the lack of relation (A,B) since it was implemented but not planned. Finally absence states the missing relation (a,b) which was not found in the code.

The evaluation is performed iteratively having as a result individual or combined modification in the architectural design, the source code or the mapping. The purposes to effectuate a static evaluation are consistency, completeness, evolution control, reuse potential, re-documentation, product alignment, component adequacy, product line potential, comprehension and structure.
For example, comprehension has the purpose of understanding the software system from a high level of abstraction by comparing planned and implementation models. Consistency worries about comparing documentation from the already implemented system and the architectural level by comparing whether

the current documentation is being updated and it is still valid or if maybe it followed different direction. Reuse potential, selected as a last purpose being exposed, evaluates the elements of the architecture to determine if these could be used within a product line. But these elements cannot be used directly, as there were not implemented for the product line, so the degree of dependencies between the desired elements and the new components determine the cost of reusing the candidate and will conduct to a decision.

Some of the benefits from static evaluation are the detection in the code of violations to the architecture, allowance of early decision about architectural problems and the possibility to communicate flaws to the architecture when they appear between architects and developers.

Static evaluations being applied are the reflexion model [MNS01], relation conformance rules [TCL04] and component access rules [Pos03].

Reflexion model requires two models: information extracted from the source code and the planned architecture. The model is called reflexion because it shows where the architecture being planned agrees with and differs from the source code model, extracted of the source code. The mapping between the high level model (planned architecture) and source code model associates entities from one model with the other. This model can be easily understood by looking again at the Figure 5.4. The high level model has the planned components and their relations. Relation is a dependency between two components. The relation can be of classified as convergence (both models match to each other), divergence (the relation only is present in the source code model) or absence (the relation only was intended and planned in the high level model). The result is the synopsis of the source code from the point of view of the high level model.

Relation conformance rules identify same errors as the reflexion model, but the mapping is performed without manually intervention. The difference relies on specifying forbidden or allowed rules between the relation of pair components.

Component access rules is based on architectural rules. Rules express conditions in multiple relations and not only one relations like the reflexion model does. The rules being used are defined using the Relation Partition Algebra (RPA). The process starts by creating architectural rules, which are formalized using RPA. Mapping associates entities between source code model and high level model. An extraction from the source code is performed and from there an abstraction of the components and relations is lifted. Verification is the next step to search for architectural violations, which are converted to source code violations. The final step presents the results.

5.4 Tools for Static Architecture Evaluation

To ensure that the code complies to the architectural design, static evaluations can be performed by the usage of tools. These tools are based in comparison principles between the real model (extracted out of the source code) and the hypothetical model (extracted from the described architecture) and will try to identify convergence, divergence or absence.

5.4.1 SAVE Tool

The Software Architecture Visualization and Evaluation (SAVE) is an Eclipse plug-in developed at Fraunhofer IESE that is described in detail in [MFK$^+$04] and [NFKM05].The SAVE tool is based on the reflexion model. The computation being done is a comparison between the source code model and the high level model (extracted out of the components, which describe the architecture) to identify differences between these two sets.

Process starts with the creation and handling of the input models (source code and high level). From there, the extraction of facts such call relations, variable usage and others is performed in automated way. The resulting information is mapped to the high level model. This information describes the relations between the source code and the components and the result is presented.
The purpose of the evaluations differ in their goals, reasons behind the evaluation and their influence in subsequent development steps.

An example of evaluation could be architecture compliance checking. Compliance checking refers to determine how adequate a specific architecture is for its intended usage and some purpose of the evaluation would be to detect violation to the relation system-architecture. For this task the tool derived some criteria for the comparison, based on the Goal-Question-Metrics approach [BCR94]. Some of the criteria being derived is listed below:

Evaluation Performance. Time required to compute the evaluation, which depends directly on the implementation and number of involved elements and their relations.

Defect Types. Describe classes of defects detected by the evaluation (i.e. broken information hiding, unintended dependencies, misuse of patterns, etc).

Maintainability. Captures the robustness of the architecture compared to the source code evolution.

Transferability. How the products being created could be reused when evaluating another version from the system.

Scalability. Captures the degree to handle large systems.

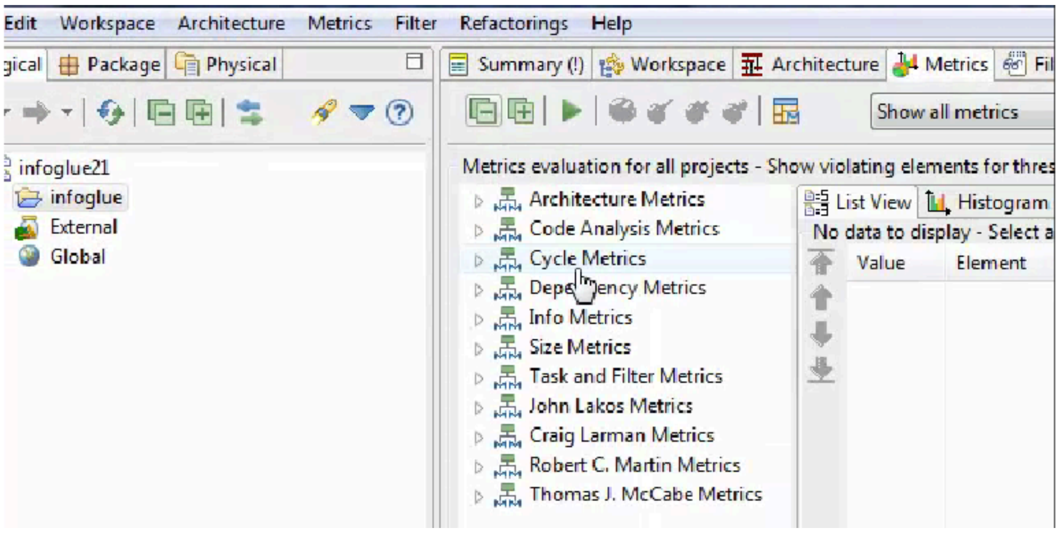
Ease of Application. Captures subjective experience from the creators of the tool composed of three levels: intuitiveness, iterations and learning curve. The first rates how easy could an analyst apply the approach going from low, medium and high; learning curve rates how much effort is required for an analyst to learn and apply an evaluation with useful results from low to high. The scale for iterations is yes and no.

Multiple View Support. There are many views from the static structure of the system, here captures how easy is to interpret several views and finding commonalities and variabilities.

The advantage of this tool during development process is the continuous architecture evaluation. The feedback about the status of the high level model and the source code model is continuous. The evaluation of architectural conformance during implementation closes the gap between early and post-deployment stages.

5.4.2 SonarJ Tool

SonarJ helps to manage and monitor the logical architecture and the technical quality of systems written in Java. It is based on static analysis and helps to find deviations between the architecture and the code. It can be used to maintain metric based software quality rules which will keep complexity under control. All the metrics are aggregated to the higher level and they can be seen in the logical architecture. Process starts by defining the logical architecture



Metrics evaluation for all projects - Show violating elements for thres

No data to display - Select a

Figure 5.5: SonarJ Tool: metrics view

and mapping it to the source code of the system, both tasks are performed manually. Next, it follows the parsing, analysis and reporting (all of them are automated tasks). During analysis, all the internal dependencies are verified and it is checked if they conform to the rules defined in the architecture. Each violation is reported. Also cyclic dependencies between packages are verified.

Most of the dependency information is gathered by analyzing the compiled class files. The reporting contains statistics, metrics for the project and a list of remaining architecture or threshold violations. Metrics being displayed cover coupling, cyclic package dependencies, size of the project and consistency problems in the architecture definition among others (See Figure 5.5).

5.5 Discussion

Architecture evaluation represent additional aid during design and development phases and it should be included in the life-cycle as a good practice. Evaluation being done during early phase from the life-cycle could show flaws or differences in the architectural design. It verifies that the requirements are fulfilled. Also the evaluation focuses in reviewing whether the systems aligns with the specification requirements. For example, evaluation can be helpful when relevant changes to the implementation (i.e. due to the evolution) could affect the architecture. Nowadays, architecture evaluation is getting more relevance in the industry and researchers are making an effort to establish and create methods to evaluate architectures, showing for example strengths and weaknesses from the system. The current techniques seem to be effective, based on the different purpose being followed. But in general, the different techniques require between mid and high level of expertise. At the moment, it does not exists an evaluation widely used, all the organizations follow their own procedures to evaluate architectures of the product, even when the techniques have similitudes among them. The experience with architecture evaluation in the industry it is presented in a lessons learned report by [Abo97].

5.6 Conclusion and related work

The paper presented different architecture evaluation methods. These methods together represent an evaluation framework that, when used appropriately, facilitates the software evolution. The architectural assessment techniques and ATAM focus on the high level implementation of the architecture and can be applied before and after the implementation phase.

Static evaluation, supported by existing tools, proves to contribute on architecture improvement and further development since it requires to understand and assesses architectural and implementation aspects. Since this evaluation is based on mapping between two models, namely planned and implemented, the comprehension of relationships between these two models and the analysis to have them looking alike can be seen as one advantage. Immediate comparison between both models can be achieved and possible violation to the architecture could be identified.

The work from this paper is related mainly to a number of research activities.

Bibliography

[Abo97] Gregory Abowd. *Recommended Best Industrial Practice for Software Architecture Evaluation*. SEI, 1997.

[BCK03] Bass, Clements, and Kazman. *Software Architecture in Practice (2nd edition)*. Addison-Wesley, 2003.

[BCR94] Victor Basili, Gianluigi Caldiera, and Dieter Rombach. *The Goal Question Metric Paradigm*. Encyclopedia of Software Engineering (Marciniak, J.J., editor), Volume 1, 1994.

[BM99] Jan Bosch and Peter Molin. *Software Architecture Design: Evaluation and Transformation*. IEEE, 1999.

[CKK01] Paul Clemens, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[IEE00] IEEE. *Recommended Practice for Architectural Description of Software-Intensive Systems*. ANSI/IEEE Std 1471-2000, 2000.

[JCJv92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-oriented software engineering. A use case approach*. Addison-Wesley, 1992.

[KKC00] Rick Kazman, Mark Klein, and Paul Clements. *ATAM: Method for Architecture Evaluation*. SEI, 2000.

[MFK$^+$04] P. Miodonski, T. Forster, J. Knodel, M. Lindvall, and D. Muthig. *Evaluation of Software Architectures with Eclipse*. IESE-Report 107.04/E, 2004.

[MNS01] Gail Murphy, David Notkin, and Kevin Sullivan. *Software Reflexion Models: Bridging the Gap between Source and High-Level Models*. IEEE, 2001.

[NFKM05] M. Naab, T. Forster, J. Knodel, and D. Muthig. *Evaluation of Graphical Elements and their Adequacy for the Visualization of Software Architectures*. IESE-Report 078.05/E, 2005.

[Pos03] A. Postma. *A method for module architecture verification and its application on a large component-based system*. Information and Software Technology 45(4), 2003.

[SEI] SEI. *http://www.sei.cmu.edu/architecture/start/definitions.cfm*.

[SF94] Joseph Sunjoe and Sisti Frank. *Recommended Best Industrial Practice for Software Architecture Evaluation*. SEI, 1994.

[TCL04] R. Tvedt, P. Costa, and M. Lindvall. *Evaluating Software Architectures*. Elsevier Science, 2004.

Chapter 6

Usability Metrics

Holger Hoffmann

Contents

Abstract: Usability metrics are used to assess various usability aspects of a software product. The evaluation can take place with users in an empirical study or without users in an automated way. The obtained values are the basis for usability reports that may suggest useful improvements or design changes.

6.1 Introduction

As software is used in more and more domains of everyday life, quality of software has become increasingly important. This is especially true for usability which is one of the critical factors for a product to be actually used and to be effective [FIA09a].

Usability metrics permit measuring certain aspects of usability in a clearly defined way. Since usability is first of all an abstract concept comprising many different facets [AI06], we must decompose usability into measurable entities. This is achieved by quality definition models for usability, also called *usability models* (see section 6.2.1). They define the required sub-characteristics. Then metrics based on cognitive science and research on Human Computer Interaction (HCI) are applied to obtain values. As in other metrics processes, these values form the basis for indicators and reports providing useful information to the product developer about how to improve the product concerning its usability.

6.1.1 Goals

The product's usability should be continuously monitored during the development process. In the early stages, models and code can be assessed. This evaluation takes place automatically *without* end users. It provides early feedback for software designers and thus allows for early improvement long before the final product becomes available.

Early evaluation also enables objective comparisons between alternate designs [IH01]. This allows the developers to choose the appropriate alternative at a time where design changes are still cheap. Since the evaluation is performed automatically we need less time for usability checks, thus waste less resources. If lacking expert knowledge in usability engineering we can still evaluate a wide range of features.

Later in the development process when an advanced prototype is at hand usability can be evaluated together *with* end users by conducting case studies. This requires quite different metrics compared to those used in an evaluation *without* users.

This paper is organized as follows: In section 6.2 different concepts of usability are introduced accompanied with the main methods for evaluating. Section 6.3 briefly describes a usability evaluation process. Sections 6.4 and 6.5 describe numerous applicable metrics in the context of some example evaluations. Finally, section 6.6 discusses current limitations, followed by the conclusions in section 6.7.

6.2 Foundations: Usability

In order to measure usability a quality definition model must be established that breaks down usability into its sub-characteristics. There are plenty of such models described in the literature [AI06]. Some are defined from scratch but most of them base upon certain ISO/IEC standards.

6.2.1 ISO/IEC standards

The most prominent among these standards are:

- ISO/IEC 9126-1 defines software quality in terms of product quality. The usability characteristic is defined to be ”the capability of the software product to be understood, learned, used, and to be attractive to the user when used under specific conditions.” [ISO01]
- ISO/IEC 9241-11 (”Guidance on usability”) is applied primarily when evaluating usability together with users. Here, usability is defined as ”the extent to which a product can be used by specified users to achieve goals with effectiveness, efficiency, and satisfaction in a specified context of use.” [ISO98]
- ISO/IEC 25000 (SQuaRE) [ISO05] is the new standard introduced in 2005 that supersedes the ISO/IEC 9126 quality definition model. It merges different existing approaches to software quality definition into one consistent model.

Based on these standards the usability models are defined in tree structures. The sub-characteristics of the particular standard are further decomposed into sets of attributes which are directly measurable [AI06]. The assessed entities and applicable sub-characteristics and metrics differ significantly depending on the evaluation type, i.e. with or without users. See section 6.4.2 for a concrete model.

6.2.2 Evaluation with users

When conducting an empirical study together with end users we are interested in usability problems related to subjective, perceptual, and conceptual issues. When using software users develop a mental model [Nor88]. By figuring out this model it is possible to detect for example potential misunderstandings [IVA07]. To evaluate with users, the final product or at least an advanced prototype is required.

This kind of evaluation focuses on external product quality and quality in a specific context of use. The data is collected from entities like log files, video observations, interviews, surveys, and questionnaires. We apply so called *testing metrics* [SDKP06] on these entities. See section 6.5 for details.

6.2.3 Evaluation without users

Evaluations without users are performed on software models or final code. In this context, *predictive* metrics are applied. They focus on internal quality attributes which influence usability. They base on a claim like: A larger font size will enhance readability and thus overall usability. [SDKP06]

The measures used are objective which is contrary to an evaluation with users which yields subjective results. For example the minimum number of steps needed to accomplish a certain task is independent of the user's preferences.

Besides analyzing the source code (e.g. Java or web content) there is a new field of research being established: *Early evaluation* proposes to perform the evaluation directly on the models used in a specific development methodology or CASE tool. In this way we may obtain preliminary results for usability very early in the development process. When interpreting these values we get a list of usability problems. These problems should be related directly to the models they originate from so the models can be improved before code generation. This method is particularly suitable for applications like database front-ends and web information systems as these applications can easily be generated from high-level models. See section 6.4 for details.

6.3 Usability evaluation process

The usability evaluation process is a typical measurement process following the "plan, do, check, act" pattern. It may consist of the following steps [FIA09b]:

1. Determine the purpose of the evaluation and the application type. De-

pending on the specific users' needs, select the relevant sub-characteristics that the metrics will focus on.

2. Specify the artifacts to be evaluated (e.g. code or models). Select the metrics with their appropriate calculation formulas. Establish some rating levels and criteria for global assessment. Determine the evaluation output by designing usability report templates, for instance a table.
3. Describe the evaluation plan, for example hand out a tasks schedule to the evaluator.
4. Execute the evaluation by applying the metrics, calculating the indicators, and generating usability reports.
5. Analyze the results and the effect of possible changes made. If not satisfied with the current usability results, select one of the proposed alternatives to improve the selected artifacts.

6.4 Usability evaluation without users

This section shows an example for early usability evaluation and the metrics used in this context. As opposed to "late" evaluation of source code early evaluation focuses on usability evaluation directly on the underlying models that are created very early in the development process.

6.4.1 Model Driven Architecture

Model Driven Architecture (MDA) is an approach to define these models and their interrelations [MDA09]. The idea is to abstract from the concrete platform and programming language to high-level concepts.

After requirements elicitation and system analysis a platform independent model (PIM) is created that contains conceptual models for e.g. the class structure and functionality. The models that are relevant for usability are mainly the navigation and presentation models.

Figure 6.1 shows how dependent on the platform used (e.g. Java) the PIM is now transformed by a model compiler into a platform specific model (PSM) and then into source code (Code Model, CM). This allows to develop the application including the user interface (UI) in a platform-independent way.

Integrating the usability evaluation directly into the MDA development process facilitates early usability evaluation directly on the constructed models [AI06]. To this end the PIM is evaluated using a usability model as described in section 6.4.2. This generates a list of potential usability problems. As these problems

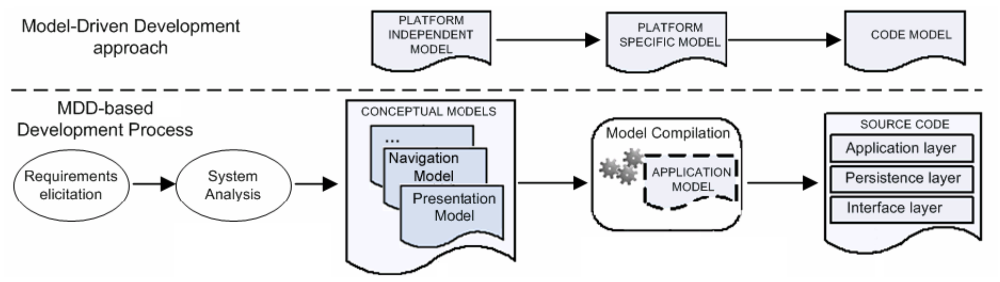


Figure 6.1: MDA development process [FIA09a]

originate directly from the PIM this list is called a *platform-independent usability report*. The same technique can of course be applied to the PSM and CM alike, generating a *platform-specific* or a *final usability report* respectively.

The reports provide the basis to improve the models. They also inform about potential problems concerning the transformation rules themselves used by the model compiler. These rules define how to translate the PIM into the PSM and how to generate code based on the PSM. These transformations may also affect the product's usability [AI06].

The whole process is depicted in figure 6.2 for a web application. The numbers (1),(2), and (3) mark the usability evaluations of the PIM, the PSM, and the CM. The procedure of evaluating and improving the models should be repeated iteratively until the severity of the usability problems detected is below some threshold.

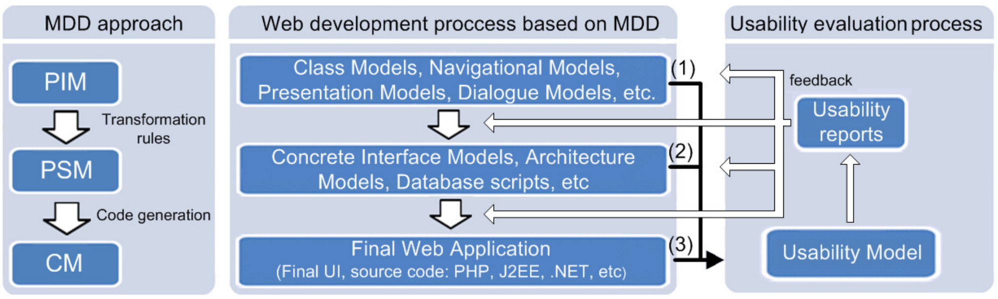

MDD approach
PIM
Transformation rules
PSM
Code generation
CM
Web development proccess based on MDD
Class Models, Navigational Models, Presentation Models, Dialogue Models, etc.
Concrete Interface Models, Architecture Models, Database scripts, etc
Final Web Application
(Final UI, source code: PHP, J2EE, .NET, etc)
(1)
(2)
(3)
Usability evaluation process
feedback
Usability reports
Usability Model

Figure 6.2: Integrating a usability evaluation process into an MDA development process for a web application [FIA09b]

6.4.2 Usability model

The usability model for early evaluation proposed by Abrahão et al. [AI06] is based on the ISO/IEC 9126-1 standard which defines five sub-characteristics:

Learnability, *understandability*, *operability*, *attractiveness*, *compliance*. These characteristics are expanded into a tree with many sub-characteristics resulting in more than 50 measurable attributes. For example the understandability sub-characteristic lists *legibility* which is further decomposed into *font size*, *contrasting text*, and *disposition*.

In order to actually use the model in an evaluation it must be *operationalized*, i.e. we have to associate quantitative and/or qualitative metrics to each model attribute. HCI research [BS93] suggests which metrics are qualified to measure which attribute. A metric definition identifies a set of variables in the respective model (PIM, PSM, or CM) and then specifies a mathematical function with these variables as parameters. The function then yields a value for the attribute. This value needs interpretation as shown in section 6.4.4.

It is observed: The higher the abstraction level of the assessed model, the less measurable usability attributes are there. The reason for this is that some attributes are only available after certain platform-dependent UI details are known [FIA09b].

6.4.3 Usability metrics

There is a vast number of usability metrics that have been developed in the last two decades [CRP05]. However it is not clear whether one of those metrics is appropriate for use in a specific evaluation context. Calero et al. [CRP05] gathered many of the known metrics and conclude that only few of them have been validated theoretically or empirically. Often there is little evidence that a metric measures the usability property we want it to measure. Moreover most metrics lack sufficient guidelines for use. This means the application of many metrics is ”difficult and risky” and it is ”dangerous to base decisions on their values” [CRP05].

The solution is to resort to a validated usability model in combination with validated metrics. Clear relationships between attributes and model variables must then be established. Finally a clear evaluation process needs to be adhered. Like all metrics, also the usability metrics should be analyzed if they meet the SQuaRE-criteria [ISO05]: There must be precise information about the metric’s purpose, interpretation, measurement method, measured artifact, and validity evidence. Particularly, a metrics definition must specify [FIA09a]:

Applicability To which model (PIM, PSM, and/or CM) can the metric be applied? Is it usable directly on the abstract models or do we need platform-specific implementation details for measurement? For example the *visibility of selections* in list boxes is determined by the window system style used so the PIM does not contain any information about that property.

Measurement method A mathematical formula that takes variables from the respective model as input parameters and yields a value on the *scale*. For example the *color contrast* between two colors C_1 and C_2 in RGB-format could be measured by

$$\sum_{i \in \{R,G,B\}} |C_1(i) - C_2(i)|$$

Scale The range of values the formula can yield. The color contrast is e.g. a real number ≥ 0. Depending on the type of the scale (categorical, ordinal, interval, or ratio scale), different mathematical constraints apply when working with these values. The color contrast is on a ratio scale, so it is allowed for example to calculate the ratio between to color contrasts.

Interpretation The correlation between the obtained value and the usability attribute. In the color contrast example this would be ”the larger, the better”. Later *ranks* can be introduced to obtain *indicators* (see section 6.4.4).

Here are two examples for usability metrics and their associations to attributes of a usability model for early evaluation:

- In order to prevent data entry errors when the user enters numerical data into a form, min/max-properties for fields should be defined in the PIM model. The metric calculates the proportion of defined min/max-properties for data entry fields [AI06]. The scale is a ratio scale with a value range within the interval [0,1], the interpretation is ”the larger, the better”, the affected usability model sub-characteristic is *operability*, the attribute is *data validity*.

- The *keystroke-level analysis* is a metric on the Code Model that measures task complexity, user workload, and long-term memory requirements [IVA07]. It counts the number of actions and time needed per action for a given task, e.g. ”add a new client to the database”. It divides the task into elementary actions (see table 6.1). This metric is difficult to automate because the individual steps must be determined manually.

Table 6.1: Keystroke-level analysis example (based on [IVA07])

1.	Recognize the ”client” label	(0.34 s)
2.	Move the mouse to the ”client details” icon and click it	(1.5 s)
3.	See the action result: A new screen shown	(0.23 s)
4.	Recognize the ”add client” label	(0.34 s)
5.	Click on ”add client”	(1.5 s)
	and so on...	

In the literature one of the prevalent application areas of usability evaluation is web application development [FIA09b] [MT07] [PCFV$^+$08a].

The traditional models for building web information systems (WIS) are *navigational maps* and a *presentation model*. The navigational map is comparable to a directed graph whose nodes represent web pages the user can see in the browser whereas edges represent clickable links between those pages. On the other hand, the presentation model defines the visual properties for information display [PCFV$^+$08a]. Data can be arranged by layout patterns like a *tabular* or a *tree*, as well as scrolling mechanisms and ordering criteria.

The usability model for early evaluation of web applications may be defined similarly to the one used when evaluating e.g. a database front-end application [FIA09b]. The following usability metrics assess the complexity of a navigational map [FIA09a]:

- The metric *breadth of a navigational map* counts the number of pages in the navigational map within the PIM model. The scale is an interval scale with values from the natural numbers, the interpretation is ”the larger the value of the metric, the harder it is for the user to understand the functionalities of the Web application.” The affected usability sub-characteristic is *user guidance* with its attribute *navigability*.
- The metric *depth of a navigational map* calculates the longest distance from the ”home” page to a leaf navigational target in the navigational map within the PIM model. The scale is again an interval scale with values from the natural numbers, the interpretation is ”the larger the distance of a leaf navigational target from the root, the harder it is for the user to reach it.” Again, the affected usability sub-characteristic is *user guidance*, attribute *navigability*.

The following metric evaluates an aspect of the presentation model.

- A web application should always use the same font for the same purpose, e.g. a heading or a link [FIA09b]. The metric *number of different font styles for textual links* counts the number of font style combinations (size, face, and color) used for these links [FIA09b]. The scale is a non-zero natural number, the interpretation is ”more than one style combination [...] means that font style uniformity is not insured” [FIA09b]. The affected sub-characteristic in the web usability model is *attractiveness*, the attribute is *font style uniformity*.

Section 6.4.4 will show how to interpret the values generated by this metric.

Molina et al. [MT07] use an approach that does not rely on a complex usability model. When measuring usability they focus on the complexity of the navigational map. They define eight types of requirements on the navigational map that can automatically be inspected by simple metrics. For instance, they allow to define a maximum distance constraint between pairs of nodes which corresponds to the number of links one has to click until reaching the other web page. Another metric measures whether two pages are directly or indirectly connected or whether one must cross certain nodes (i.e. pages) when one wants to reach a target page (for example a log-in page must be shown prior to reaching a page displaying private messages). However, these constraints must be defined manually and for each single application.

6.4.4 From values to decisions

All the presented metrics yield some numerical or categorical values on a defined scale but up to now the value's meaning is unclear. Therefore the next step is to introduce indicators that interpret the values. The combined indicators create an information product and thus a base for decisions. [ISO07]

For instance consider the metric *depth of a navigational map* from the previous section. HCI research indicates that the depth of such a navigational map should be lower than 5 to be acceptable [FIA09a]. In this case the indicator for the attached usability attribute is "good" or "no usability problem". Values of above 4 lead to "bad" or "usability problem detected". In this case the usability report will contain an entry for this problem (see section 6.4.5).

Apart from boolean values, the indicator can also be on an extended qualitative scale. The metric that counts the number of different font styles for text links in web applications (section 6.4.5) is mapped to the following *ranks* [FIA09a]:

Value	Rank
1	no usability problem
2	low usability problem
3	medium usability problem
> 3	critical usability problem

A different way of defining ranks is to use a generic scale like (very good, good, medium, bad, very bad) [PCFV$^+$08a]. For each metric we can define intervals that map values of the metric's scale to these 5 indicators. Thus we can automatically construct a global usability result in a simple way.

The indicators for attributes of the usability model are now combined to form indicators for a sub-characteristic, and these are again combined for overall usability. This can be done e.g. by a rewriting terms scheme [PCFV$^+$08a]. Combining "very good" and "very bad" yields "medium", for example, and "good" and "good" yield "very good". That is similar to term rewriting rules:

$$\begin{array}{ll} \text{Combination(VG,VB)} & \rightarrow \text{M} \\ \text{Combination(G,G)} & \rightarrow \text{VG} \end{array}$$

This way of indicator combination came under criticism during empirical studies [PCFV$^+$08b]. Simply averaging the indicators does not necessarily reflect the end users' perception as negative usability is more significant to users than positive. In other words, bad results prevail.

Another problem arises if the ranks of the indicators are too strict: If there is only one step between good and bad the evaluation results do not correlate well with the evaluation with users because of too high variance [PCFV$^+$08b]. The authors of the study propose instead a "fuzzy logic" or at least a somewhat finer indicator scale.

Nevertheless indicator values with only a few ranks are expedient. They work as a "traffic light" so the evaluator quickly gets a hint where usability problems are located and where to improve the application.

6.4.5 Usability report

A usability report contains the usability indicators for all the sub-characteristics and attributes. It also includes a list of the usability problems found. The format of such a report was defined during the evaluation process schedule (see section 6.3). Table 6.2 shows an example for an entry in the report. In this case there were multiple font styles used for links in a web application which affects the *attractiveness* sub-characteristic with its attribute *font style uniformity*.

Table 6.2: Example for an entry in the usability report (based on [FIA09b])

Usability problem ID	UP001
Description	The links A,B,C are displayed in a font style that is different from the font style of the D link.
Affected attribute	Attractiveness / font style uniformity
Level	Low (rating level: 2)
Source of the problem	Presentation Model (PIM model)
Occurrences	2 occurrences (top menu and left menu)
Recommendations	Change the font style properties for the links A,B,C in the Presentation Model.

6.5 Evaluation with users

In the later stages of the development process, when the final product or at least an advanced prototype is available, the focus of usability evaluation shifts to empirical studies together with end users. As already mentioned in the foundations section these studies can reveal usability problems related to subjective perception.

The first step within such an empirical study is to prepare the evaluation means, e.g. design interviews, surveys, and questionnaires. All means must meet some special requirements that make sure the obtained results will be significant. These requirements originate from descriptive statistics and are common to psychological studies: The usability model must be checked for *validity* and *reliability* [PCFV$^+$08a] so the applied metrics will yield values that represent the user's actual perception and opinion.

There are many usability models especially for evaluation *with* users [SDKP06]. Seffah et al. [SDKP06] tried to unify the available models by proposing a usability measurement model called Quality in Use Integrated Measurement (QUIM). It consists of 10 factors (efficiency, effectiveness, productivity, satisfaction, learnability, safety, trustfulness, accessibility, universality, usefulness), 26 sub-factors or measurable criteria, and 127 *testing* or *predictive metrics*.

When evaluating usability with users, no predictive metrics are required since the evaluation assesses the actual usability perceived by the user. So only *testing metrics* are applied. They can be classified as [SDKP06]:

Preference metrics that quantify subjective evaluations, preferences, and the level of satisfaction of end users or

Performance metrics that measure the actual performance of users when accomplishing a task in a certain context.

6.5.1 Example evaluation

The following example delivers insight into a usability evaluation of a web application within an empirical study together with end users [PCFV$^+$08a]. The goal of the study was to examine the difference between the user's perceptions of usability and the results from the early evaluation method.

At first, the users had to fill out a demographic questionnaire asking for gender, education level, and professional experience. Then the participants got a list of tasks they had to accomplish using that application they never used before. If a user got stuck he or she could ask for assistance. The *testing metrics* used in the study were based on video tapes and questionnaires.

The video recording informed about about the *number of assists*, the *time* needed, and *error* and *completion rates* per task. The perception-based variables were retrieved by a questionnaire using a scale with 5 checkable points from very good to very bad. The questions asked for *screen factors*, *termi-*

nology and *system feedback*, *learning factors*, *system capabilities*, *satisfaction*, and a global variable for *overall user reaction*. Furthermore, there were open questions, i.e. questions that offer the possibility to give detailed comments. For example, the users were asked to ”list the three things” that they ”least/most liked in the application” in order to quickly determine the parts of application needing a modification.

The results for usability evaluation with users can now be compared to the results when evaluating without users. As the next section will show this usually reveals some limitations of early usability evaluation techniques.

6.6 Limitations

The study of Panach et al. [PCFV$^+$08b] shows there can be significant differences between the internal usability measures from early evaluation and, on the other hand, external measures such as the perceptions of the study participants. The authors found that internal measures reflect only a portion of the total usability measures, i.e. they are only a prediction of the system usability. Many attributes can only be measured when the final system is available.

User preferences and misconceptions cannot be measured without manual procedures such as user tests and interviews [IH01]. So when using automatic evaluation only we cannot capture important qualitative and subjective information. This may explain that internal attributes do not always represent the usability perceived by the end-user [PCFV$^+$08b].

In general, a usability evaluation may only cover a subset of possible user actions. Moreover users are quite heterogeneous: Learnability, for example, is mainly an issue for novice users. Evaluations can hardly give a universal result for the usability of a system since there is no way to please all users [IVA07]. The solution to these problems is combining several different techniques such as early, late, and user-centered evaluation [IH01]. Furthermore, the usability models should be customizable to different user requirements [AI06].

Most related studies show how the usability of a small example application is evaluated within a research project in cooperation with some university [IVA07] [FIA09b] [PCFV$^+$08b]. However, papers are missing describing the application of automatic usability evaluation without users in a larger scale. This applies in particular to industry applications.

Finally, there are various stand-alone tools that have been developed by researchers to support the evaluation process like usability model editors and automatic calculation of metrics values and indicators from models [MT07] [IH01]. In most cases they are still waiting for integration in professional CASE tools.

6.7 Conclusions

This paper presented various techniques for continuous monitoring of software usability during the development process. The author illustrated a usability evaluation process (section 6.3) and usability models. He stated the peculiarities of evaluations without and with users, and the applicable metrics in both cases.

The paper showed that early evaluation of model-generated applications is useful during the early stages of the development process as this helps to guide developers in finding potential usability problems. Once the usability model is set up and all model transformation rules have been defined, the usability of the final product can be estimated within an automatic process.

However, early evaluation is not a substitute for evaluations together with end-users when reaching the final product [IH01]. Empirical evaluation remains important. Best results can be achieved when combining the existing methods and using them along the development cycle when appropriate.

According to the authors of many articles further research is required in many areas of usability evaluation. The metrics need to be theoretically or empirically validated and to be inserted into a detailed metrics catalog. Results of evaluations without users and findings of empirical studies measuring similar usability aspects should better correlate. Finally the aim is to gain experience within large industry projects and to appraise to what extent the presented techniques are indeed helpful to improve the product and to reduce costs.

Bibliography

[AI06] S. Abrahão and E. Insfran. Early usability evaluation in model driven architecture environments. In *Quality Software. QSIC 2006. Sixth International Conference on*, pages 287–294, 2006.

[BS93] J. Bastien and D. Scapin. Ergonomic criteria for the evaluation of human-computer interfaces, version 2.1. Institute National de Recherche en Informatique et en Automatique (INRIA), 1993.

[CRP05] C. Calero, J. Ruiz, and M. Piattini. Classifying web metrics using the web quality model. *Online Information Review*, 29(3):227–248, 2005.

[FIA09a] A. Fernandez, E. Insfran, and S. Abrahão. Integrating a usability model into model-driven web development processes. In *Web Information Systems Engineering - WISE 2009*, volume 5802/2009, pages 497–510. Springer-Verlag LNCS, 2009.

[FIA09b] A. Fernandez, E. Insfran, and S. Abrahão. Towards a usability evaluation process for model-driven web development. I-USED'09 Upssala, Sweden, 2009.

[IH01] M. Ivory and M. Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, 2001.

[ISO98] *ISO 9241-11, Ergonomic requirements for office work with visual display terminals - Part 11: Guidance on Usability*, 1998.

[ISO01] *ISO/IEC 9126-1, Software engineering - Product quality - Part 1: Quality model*, 2001.

[ISO05] *ISO/IEC 25000, Software Product Quality Requirements and Evaluation (SQuaRE)*, 2005.

[ISO07] *ISO/IEC 15939, Systems and software engineering – Measurement process*, 2007.

[IVA07] E. Iborra, J. Vanderdonckt, and S. Abrahão. Usability evaluation of user interfaces generated with a model-driven architecture tool. In *Maturing Usability: Quality in Software, Interaction and Value. HCI Series*, pages 3–32. Springer-Verlag, 2007.

[MDA09] MDA. `http://www.omg.org/mda`, 2009. accessed 2009-12-10.

[MT07] F. Molina and A. Toval. A generic approach to improve navigational model usability based upon requirements and metrics. In *Web Information Systems Engineering - WISE 2007*, volume 4832/2007, pages 511–516. Springer-Verlag LNCS, 2007.

[Nor88] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 1988.

[PCFV$^+$08a] J. Panach, N. Condori-Fernández, F. Valverde, N. Aquino, and O. Pastor. Towards an early usability evaluation for web applications. In *Software Process and Product Measurement*, volume 4895/2008, pages 32–45. Springer-Verlag LNCS, 2008.

[PCFV$^+$08b] J. Panach, N. Condori-Fernández, F. Valverde, N. Aquino, and O. Pastor. Understandability measurement in an early usability evaluation for model-driven development: an empirical study. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 354–356, New York, NY, USA, 2008. ACM.

[SDKP06] A. Seffah, M. Donyaee, R. Kline, and H. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.

Chapter 7

Requirements Quality

Andrea Hutter

Contents

Abstract: Measuring a software requirement specification is an important aspect in software development. Hereby it is possible to improve the requirement specification and to provide the development workload. This paper presents and analyzes different measurement techniques for requirement documents. Different methods for quality and size measures like inspections or the function point analysis are introduced. Furthermore existing tools to support these measurement techniques are expounded.

7.1 Introduction

Nowadays, many technical devices rely on correct working software, since malfunctions would render the whole device useless. Many standards are developed to assure this functionality by the International Organization for Standardization (ISO), the Institute of Electrical and Electronics Engineers (IEEE) or other organizations. One of the most important standard for software development is the ISO/IEC 12207 [ISO08], a standard that defines the software lifecycle process. It is divided into different parts like development and maintenance, documentation or management. This paper describes an important part of the development process: measurement of requirement engineering, especially software requirement specifications.

To create a new software system, requirement engineers have to specify all important requirements of the system. There are various functional and non-functional requirements like technical or design requirements. To get an overview of the software system it is important to collect all of these requirements in a software requirement specification (SRS) that considers different documentation types. This SRS includes all mandatory functionalities, requirements and system attributes for the software system and therefore it is mostly used as a binding contract between customer and supplier. An understandable, readable and unambiguous specification is therefore essential. Several metrics can be defined to evaluate the given requirement specification. These can give an overview of potential defects in a SRS and its quality (see 7.2.3). They may also be used to predict the complexity of the development process.

In this paper we give an overview of several metrics and will describe and evaluate them. First of all fundamental informations are defined and described in a background section. In section 7.3 we present the general idea of the whole measurement process and two different measurement classes are identified: quality and size measurement. In the following section measuring tools are presented. Finally we give a short discussion about the given measures, an outlook to other methods and conclude the topic.

7.2 Background

This paper deals with the requirement engineering process and how to measure its quality, therefore the following section will give background information on the requirement engineering process, the main idea of software requirement specification qualities and the definition of measurement will be clarified. Finally, the question ”why it is important to measure?” is addressed.

7.2.1 Requirement Engineering Process

The requirement engineering process includes conceptualizing, specifying and validating the requirement specifications of a developing software system. Its goal is to produce a stable, traceable and complete set of requirements [CL95]. This process is a part of the whole system development process and very important for the later development. Hence, the results should be a high-qualitative software requirement specification (SRS) without any defects.

7.2.2 Software Requirement Specification

The SRS is not only a collection of requirements but also a contract between the system supplier and the customer. It can be composed by stakeholders from both sides and it should provide all information that are necessary to build the system and to draft the contract. The Institute of Electrical and Electronics Engineers defined the SRS in the IEEE Std-830-98 [IEE98] as ”a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment”. They defined the structure of the SRS as follows:

- **introduction** (contains purpose, scope, definitions, references and overview of the SRS)
- **overall description** (includes product perspective and functions, user characteristics, constraints, etc.)
- **specific requirements** (all kind of requirements)
- **supporting information** (like table of content, appendixes and an index)

As mentioned above there are different types of requirements. The top level derivations are functional and non-functional requirements.
A **functional** requirement describes all functions of a system which results can be seen during the system operation. Therefore the in- and output is important for a functional requirement. Other requirements like agility, maintenance or

design aspects are of no interest at this point. Functional requirements can be specified in Use Cases and they can be measured with function points (see 7.3.2).
A **non-functional** requirement describes constraints which restrict the system functionality (e.g. design (-draft), requirement to process, quality requirements). Non-functional requirements are normally written in natural language and can be proved and measured by linguistic analyses and reviews (see 7.3.1).

A SRS can be written either in formal or informal language. It is common to use the English language in a less formal way to assure understandability between all SRS users. Nevertheless, there are some drawbacks given by a less formal language as presented in 7.2.5.

Different types of documents like function documents, tables, modeling diagrams or Use Cases exist in a SRS. Use Cases are of great significance in this paper so we will define them according to [LL07].

Use Case Use Cases (UC) describe system functionality from the user's point of view. The interaction between the system and its environment is defined. Therefore at least one actor (user or external system) triggers an event to reach a goal. A UC is goal-oriented and describes a concrete system behavior with pre- and postconditions, the normal and special cases to reach this goal. Normally the textual description is written in natural language, but it is structured formally. It can be visualized in an UML Use Case diagram.

A SRS should fulfill a high quality standard to secure a good development process. Therefore we will define requirement qualities in the next section.

7.2.3 Requirement Qualities

A high quality SRS "contributes to [a] successful, cost-effective creation of software that solves real user needs" [DOJ+93]. To reach this goal there are several quality aspects which should be followed. We present the most important aspects according to IEEE Std. 830-1998 [IEE98]:

Complete A SRS is complete if and only if it contains every system requirement and no system functionality is left; every reference to tables, diagrams, figures and pages are set and there is no use of the phrase "to be determined" (TBD).

Consistent A SRS is consistent if and only if no requirement stated therein conflict with another requirement of the SRS or any other document of the system plan.

Correct A SRS is correct if and only if every requirement describes a condition of the system.

Modifiable A SRS is modifiable if and only if changes in the specification can be done easily, consistently and completely.

Ranked for importance and/or stability A SRS is ranked for importance/stability if and only if the requirements are labeled by an importance/stability tag that every reader can detect a ranked order easily.

Traceable A SRS is traceable if and only if the reference and origin of each requirement is clear and easy to find.

Unambiguous A SRS is unambiguous if and only if there is only one way to interpret each requirement.

Verifiable A SRS is verifiable if and only if every requirement can be verified

7.2.4 Measure, Measurement, Metric

Several definitions for the word ”measure” exist. This section defines the terminology for this paper according to ISO/IEC 15939 [ISO07].

base measure A base measure is a single ”attribute and the method for quantifying it”. All base measures are functionally independent. In various literature (e.g. [FP91]) this measure is called ”metric”. In this paper a base measure is also called metric to stress the difference between base and derived measure.

derived measure A derived measure is a function of two or more metrics. We will call it measure.

measurement A measurement is a ”set of operations having the object of determining a value of a measure” or metric.

The aim of measurement in requirement engineering is to explain the defects that occur in a SRS, to give an overview of its quality and to predict the project progress. Therefore it is important to find a scale and create a framework to compare different metrics and measures because a single metric/measure has no meaning otherwise.

7.2.5 Drawbacks

After defining the aim of measurement, we will expose the drawbacks of a non-evaluated SRS. As mentioned in 7.2.2 the SRS is a contract between supplier and customer. Therefore a high qualitative SRS is important, else high costs in rework or even a refusal of the system by the customer might happen. To prevent high costs a SRS measurement can be performed because most of the defects arise in requirement engineering. Whether all quality aspects like understandability or unambiguity, which are caused by the use of natural language, are kept can be proven by the use of metrics. It is also possible to draw conclusions regarding the development process by comparing measurement values of the given specification to those of earlier projects. Cumulatively adding metrics can improve the whole development process.

7.3 Measurement Process

Defined in ISO/IEC 15939 [ISO07], a measurement process is a ”process for establishing, planning, performing and evaluating measurement with an overall project, enterprise or organizational measurement structure”. The process of performing and evaluating measurement is explained in this section. We will focus on two different measurements for SRS: quality and size. First of all, the quality measurement with its representatives review and linguistic analysis will be presented. In the second part of this section the size measurement will be explained on the basis of the function point analysis and the Use Case analysis.

7.3.1 Quality Measurement

The first measurement aspect presented in this paper is the quality measurement. To increase the quality (as defined in 7.2.3) of a SRS the documents have to be verified before reaching the next development phase. Therefore various techniques exist to build up metrics that give an overview of the SRS quality. These techniques and how to interpret the given metrics are described in this section.

Review and Inspection

A review is often used to measure specific requirement quality aspects. Reviews are a ”specific document examination technique” [SLS07] which is done manually. Several types of reviews exist like the technical or informal review, the walkthrough and the most formal review type: the inspection, which is used in this paper. The inspection process includes several persons who are involved in the actual development. In general, the goal of an inspection is to find defects as soon as possible after completing the SRS and decrease rework costs in a later development stage. More than 70% of the defects can be found during an inspection session which costs between 10% to 15% of the whole budget. [SLS07]

Several SRS qualities given in 7.2.3 like completeness, correctness, verifiability or (external) consistency can be measured by reviewing the specification documents. In table 7.1 some measures are given according to [DOJ$^+$93]. Completeness is measured by the percentage of requirements that are well understood. Therefore every requirement that is well understood by every reviewer is counted (n_u) and this value is divided by the number of all requirement. Correctness can be measured in a similar way: the metric is the percentage of requirements that are correct. To reach a high quality of the SRS both values should approximate to 100 %.

QUALITY NAME	QUALITY MEASURE	DESCRIPTION
completeness	$\frac{n_u}{n}$	percentage of requirements that are well understood (n_u) with n number of all requirements
correctness	$\frac{n_c}{n}$	percentage of requirements that are correct (n_c) with n as above

Table 7.1: Example for non-functional quality measures determined by an inspection

Certain techniques to detect defects in a SRS are available. On the one hand, non-systematic methods like the ad hoc-method (review the document without any restriction) or a checklist (with all important questions, learned in earlier inspections) are given. On the other hand, there are systematic methods that improve an inspection process like defect-based-reading, perspective-based-reading and metric-based-reading (MBR). In this paper only the last method will be presented, the others may be found in [PVB95] or [BGDT04].

The MBR is a technique based on a set of heuristics that detects defects in a Use Case review. The MBR-process is a reading and counting method in which a number of certain UC factors are counted and evaluated. To proof a good written UC the measure values should be in a particular range. If they are not, the document would have to be revised. In table 7.2 the Use Case measures and their ranges are given according to [BGDT04] with μ = expected value and σ = standard deviation. The first measure is the NOS, which describes the number of steps in a Use Case. Usually a UC contains four to nine steps. NOAS describes the number of actor action steps, so the NOAS/NOS measures define the rate of actor action steps which normally is between 30% and 60%. The rate of system action steps (NOUS/NOS) ranges normally between 40% and 80% and the rate of Use Case actions of a Use Case is usually between 0% and 35%.

MEASURE NAME	MEASURE VALUE	DESCRIPTION	USUAL RANGE
NOS	$\in [0, \infty]$ $(\mu = 5.70, \sigma = 2.64)$	Number of steps of the UC	$[4, 9]$
NOAS/NOS	$\in [0, 1]$ $(\mu = 34.52\%, \sigma = 17.74\%)$	rate of actor action steps of the UC	$[30\%, 60\%]$
NOSS/NOS	$\in [0, 1]$ $(\mu = 59.71\%, \sigma = 18.80\%)$	rate of system action steps of the UC	$[40\%, 80\%]$
NOUS/NOS	$\in [0, 1]$ $(\mu = 5.77\%, \sigma = 10.42\%)$	rate of UC action steps of UC	$[0\%, 35\%]$

Table 7.2: UC measures for MBR

Linguistic Analysis

Besides the mentioned defects linguistic problems often occur while writing the SRS. The use of natural language is often preferred so non-experts may understand it easily. Therefore numerous SRS are written in plain English (or any other language), but this causes problems. Most languages are ambiguous themselves so a proper and unique interpretation of a requirement is often difficult. It is possible to introduce a formal language to handle this problem, but as a consequence project members and even customers (who are probably not content to do so) have to be instructed in applying it. Accordingly the SRS has to be proven for specific quality indicators which can be measured and interpreted.

Writing in natural language causes certain defects like omission of words or important information by doing implicit assumption or using superlatives. [Rup07] Also, humans often tend to generalize requirement statements by using universal quantifier. Therefore checking the SRS systematic for these kind of defects is necessary. This is often done by reviewers with high linguistic and analytic experience. On the other hand, several quality indicators can be analyzed automatically. It is possible to determine the size of the document, number of used imperatives, weak phrases or directives like figures, tables or notes [WRH97]. These values are metrics of the SRS which can be evaluated, e.g. the higher the number of weak phrases, the worse is a unique interpretation of it. Whereas counting these metrics takes a long time, there are tools available capable of measuring and evaluating them. These tools will be shortly presented in section 7.4.

7.3.2 Size Measurement

Improving the quality of a SRS is not the only purpose to measure. Different methods to evaluate the specification in the whole development content exist as well. It is possible to give a report on its complexity and predict its workload. These methods are presented in this section.

Function Point Analysis

For measuring ”the amount of functionality in a system as described by a specification” [FP91] from the user’s point of view, the Function Point Analysis (FPA) can be applied. Furthermore the complexity of the development process can be assessed by comparing the given function points (FP) to ones of earlier projects. The FPA was developed by Allan Albrecht of IBM in 1979 and was standardized later by the International Function Point Users Group (IFPUG) which turned the subjective FPA process into an objective one.

The FPA process can be described as follows:

1. Related groups of data like internal logical or external interface files and external inputs, outputs and inquiries should be identified.

2. The unadjusted function points (UFP) can be calculated as follows: counting all data and record element types (DET, RET) of each data group and value their complexity as low, average or high (table 7.3). (e.g.: 19 DET and 2 RET lead to a low complexity ”L”, 23 DET and 7 RET lead to a high complexity ”H”)

RECORD ELEMENT TYPES	Data Element Types (DET)		
(RET)	1 TO 19	20 TO 50	51+
1	L	L	A
2 to 5	L	A	H
6+	A	H	H

Table 7.3: Complexity of data type functions according to [AR96], table 5

3. These values are accumulated according to Albrecht’s weight table from 1983 (given in table 7.4) to get the UFP. (e.g. an internal logical file with an average complexity with the value 10 + an external interface file with a low complexity (5) + ...)

FUNCTION TYPE	LOW	AVERAGE	HIGH
Internal logical file	7	10	15
External interface file	5	7	10
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6

Table 7.4: Weights of function types according to [AR96], table 2

4. The technical complexity factor (TCF), which consist of 14 technical complexity factors that can deviate the UFP with a deviation about $\pm$ 35%, has to be defined ([FP91], table 10.3).
5. To receive the actual function point value (FP), the calculation of following equation is necessary: FP = UFP $\times$ TCF

A scale to rate this measure is necessary to define. Therefore it is possible to compare this value with FP of other projects in order to evaluate the size and complexity of the actual project.

Albrecht's FPA was developed in 1979. Therefore new function point methods adapted to new development technologies were necessary. An existing technique is the COSMIC Method (see [TA08]).

Use Case Points

FPA takes no consideration in new specification methods. Therefore another kind of point analysis that also includes modern specification techniques like UML, the Use Case Points method (UCP), exists. In general the application is similar to FPA but there are different modifications. To get the unadjusted UCP it is necessary to assign weights and complexities of any Use Case to the specification. Additionally the actors get classified according to their type and this value is added to the total amount. Now the unadjusted Use Case Point (UUCP) is calculated. Similar to the FPA there are various factors that can disturb the real value. Hence, one has to add a technical complexity factor (TCF) (with 13 influential factors, TCF$\in [0.6 - 1.3]$ and a environmental factor (EV) (with 8 influential factors and EV$\in [0.425 - 1.7]$). At last we receive the adjusted UCP by calculating following equation: UCP = UCP $\times$ TCF $\times$ EC. It is also possible to predict the process time by adding an additional productivity factor (PF) according to the productivity of the organization to the UCP [FJE06].

7.4 Tools

Tools can be a significant support for measuring software, especially SRS. Usually specification documents comprises several pages, therefore a manual measurement for all aspects is difficult. All measures can be obtained by a manual measurement. Often there are many aspects to count and control, so it is more comfortable to use an automated tool. Nevertheless, not every measurement can be done automated. In this section the usage of tools for the presented measurement processes in section 7.3 are introduced.

Inspection An inspection is a typical manual technique. Working experience of all participants is an important factor which cannot be replaced by a tool. It is only possible to use administration tools or knowledge databases. Administration tools can support the whole review process by collecting all necessary information like participants, dates, documents and inspection rules. Knowledge databases collect the knowledge of previous inspections like findings or difficulties in the inspection process. These informations can be used to optimize the review processes.

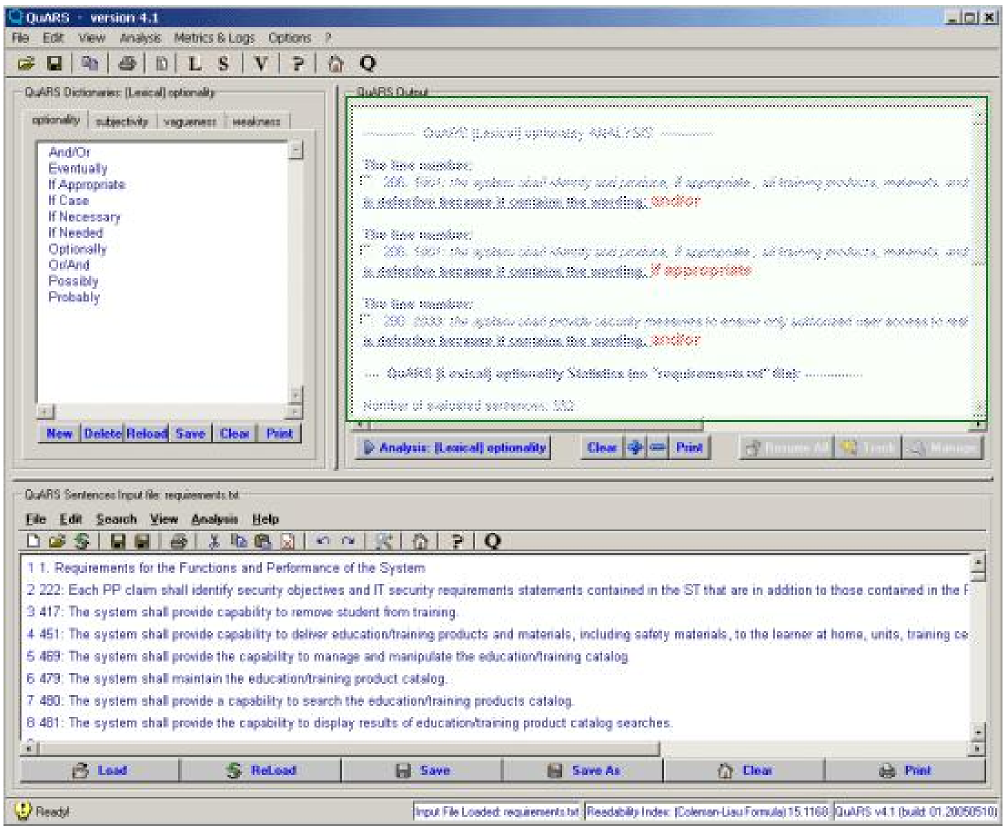
Linguistic Analysis Two sides of a linguistic proof exist: the manual and the automated analysis of the SRS. The first is described in 7.3.1. For the second kind of linguistic proof various tool are available. We will present two of them.

QuARS
The Quality Analyzer of Requirement Specifications (QuARS) based on a quality model which verifies the following quality properties: non-ambiguity, specification completion, consistency and understandability [FFGL01]. As mentioned in 7.3.1 there are several indicators for analyzing a SRS. Table 7.5 gives an overview of some indicators and its keywords detected by QuARS. Non-ambiguity has several indicators like vagueness or weakness. A requirement is vague if it points out words which have a non-uniquely quantifiable meaning (e.g. clear, easy, significant). A sentence containing a weak main verb (e.g. could, might) is weak. An indicator for understandability is implicity. A requirement is implicit if a subject in this sentence is generic rather than specific (e.g. demonstrative adjectives like ”this”, ”that” or prepositions like ”above”, ”below”). Counting these indicators can indicate the strength or weakness of a SRS.

PROPERTY	INDICATION	DESCRIPTION	KEYWORD
non-ambiguity	vagueness	pointing out words having a non uniquely quantifiable meaning	e.g.: clear, easy, useful, significant, ...
non-ambiguity	weakness	sentence containing a weak main verb	e.g.: could, might, ...
...	...	...	...
understandability	implicity	subject in a sentence is generic rather than specific	e.g.: demonstrative adjectives (this, these, that,...), preposition (above, below,...), ...

Table 7.5: Example for QuARS identifiers referring to [FFGL01]

QuARS parses the SRS and gives warning messages for every sentence in which potential defects appear but does not correct them. This arises the tool to a helpful support for checking specification documents without any restriction. Furthermore it can be adapted to different domains by expanding the checking dictionary for domain-specific vocabulary. Figure 7.1 shows the user interface of this tool. On the left side there is a dictionary frame with QuARS and domain specific dictionaries. The input frame on the bottom includes the SRS and the output of QuARS is given in the output frame on the right side. More information about QuARS are given in [FFGL01].


QuARS - version 4.1
File Edit View Analysis Metrics & Logs Options ?
QuARS Dictionaries: [Lexical] optionality
optionality subjectivity vagueness weakness
New Delete Reload Save Clear Print
QuARS Output
Analysis: [Lexical] optionality
Clear Print
Resume All Track Manage
QuARS Sentences Input file: requirements.txt
File Edit Search View Analysis Help
Load ReLoad Save Save As Clear Print
Ready!

And/Or
Eventually
If Appropriate
If Case
If Necessary
If Needed
Optionally
Or/And
Possibly
Probably

```
------ QuARS [Lexical] optionality ANALYSIS ------

The line number:
[illegible]
is defective because it contains the wording: and/or

The line number:
[illegible]
is defective because it contains the wording: if appropriate

The line number:
[illegible]
is defective because it contains the wording: and/or

... QuARS [Lexical] optionality Statistics (on "requirements.txt" file) ..........

Number of evaluated sentences: [illegible]
```

```
1 1. Requirements for the Functions and Performance of the System
2 222: Each PP claim shall identify security objectives and IT security requirements statements contained in the ST that are in addition to those contained in the F
3 417: The system shall provide capability to remove student from training.
4 451: The system shall provide capability to deliver education/training products and materials, including safety materials, to the learner at home, units, training ce
5 469: The system shall provide the capability to manage and manipulate the education/training catalog
6 479: The system shall maintain the education/training product catalog.
7 480: The system shall provide a capability to search the education/training products catalog.
8 481: The system shall provide the capability to display results of education/training product catalog searches.
```

Input File Loaded: requirements.txt | Readability Index: (Coleman-Liau Formula) 15.1168 | QuARS v4.1 (build 01.20050510)

Figure 7.1: Screenshot of QuARS user interface [QuA]

ARM
The Automated Requirements Measurement software (ARM) was also developed for parsing requirement specifications. Originally, it was used for NASA projects in order to strengthen their requirement specifications. This tool is not supposed to an ”evaluation of correctness of a specified requirement document”, but to provides measures for an appraisal of its quality [FGLM03]. The ARM quality model is similar to the model of QuARS with additionally indicators for individual NASA requirement specifications. Its output is a data collection which can be evaluated by the user. We tried to install and use ARM, but an error occurred while we tried to load a SRS. The support of this tool stopped in January 2004. Further information can be found in [WRH97], [FGLM03] or on the official Website [ARM].

FPA and UCP Functional size measurement techniques are mostly used manually by an analyst. Nevertheless a tool like ”Function Point WORKBENCH” [FPW] exists, with support for counting, analyzing and documenting function points.

7.5 Discussion

Measurement is essential for the whole development process. Hence, the question if it is possible to use every measure for every development domain arises. We have to deny this, it is important to adapt most measure processes to the specific project. Therefore it is not always easy to find the right metrics. Additionally it is important to define a scale and a framework for the given measures. A single measure does not allow to draw conclusions, it is necessary to link metrics to an evaluation framework.

Measures give only a general idea of the SRS quality. Proving its accuracy and correctness is the main task for the requirement engineer. Therefore a working understanding of requirement engineering, measurement processes, the given language and evaluating the measurement results is important for every engineer involved in the specification development and measuring process.

7.6 Outlook

Plenty of other measures and measurement processes exist which were not presented in this paper. It might also be interesting to consider process metrics for evaluating the whole development and managing process. Furthermore some techniques are obsolete for new domain specific development. Therefore a method exists to find new suitable metrics called the Goal-Question-Metric [KA97] which is usually used to find new metrics. It might be interesting to analyze the effectiveness and the adaption of measures to requirement specifications of different domains.

7.7 Conclusion

This paper gave an overview of the importance of measuring the SRS of a software project. The measure values can be appraised and the specification can be improved consequently on the basis of these values. Furthermore it is possible to assess the systems complexity and predict the following workload, hence it is possible to give the customer an estimation of costs. After all, measures are only supporting tools to improve quality. To receive a high quality SRS the experience and motivation of the involved requirement engineers is essential.

Bibliography

[AR96] A. Abran and P.N. Robillard. Function points analysis: an empirical study of its measurement processes. *Software Engineering, IEEE Transactions on*, 22(12):895–910, Dec 1996.

[ARM] http://satc.gsfc.nasa.gov/tools/download/. 08.02.2010.

[BGDT04] B. Bernardez, M. Genero, A. Duran, and M. Toro. A controlled experiment for evaluating a metric-based reading technique for requirements inspection. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, pages 257–268, Sept. 2004.

[Che08] Ting Chen. The application of the function point analysis in software developers' performance evaluation. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4, Oct. 2008.

[CL95] Rita J. Costello and Dar-Biau Liu. Metrics for requirements engineering. *Journal of Systems and Software*, 29(1):39 – 63, 1995. Oregon Metric Workshop.

[DOJ+93] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium, 1993. Proceedings., First International*, pages 141–152, May 1993.

[FFGL01] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard*, pages 97–105, 2001.

[FGLM03] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Applications of linguistic techniques for use case analysis. *Requirements Engineering*, 8(3):161–170, Aug. 2003.

[FJE06] Stephan Frohnhoff, Volker Jung, and Gregor Engels. Use Case Points in der industriellen Praxis. In *Applied Software Measurement - Proceedings of the In-ternational Workshop on Software Metrics and DASMA Software Metrik Kongress*, pages 511–526. Shaker Verlag, 2006.

[FP91] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 1991.

[FPW] http://www.charismatek.com.au/_public4/html/fpw_brochure.htm. 08.02.2010.

[IEE98] *IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications*, 1998.

[ISO07] *ISO/IEC 15939:2007, Systems and software engineering – Measurement process*, 2007.

[ISO08] *ISO/IEC 12207:2008, Systems and software engineering – Software life cycle processes*, 2008.

[KA97] Tereza G. Kirner and Janaina C. Abib. Inspection of software requirements specification documents: a pilot study. In *SIGDOC '97: Proceedings of the 15th annual international conference on Computer documentation*, pages 161–171. ACM, 1997.

[LL07] Jochen Ludewig and Horst Lichter. *Software Engineering*. dpunkt, 1. aufl. edition, 2007.

[PVB95] A.A. Porter, Jr. Votta, L.G., and V.R. Basili. Comparing detection methods for software requirements inspections: a replicated experiment. *Software Engineering, IEEE Transactions on*, 21(6):563–575, Jun 1995.

[QuA] http://quars.isti.cnr.it/images/gs06.gif. 07.02.2010.

[Rup07] Chris Rupp. Requirements Engineering und Management. Professionelle, iterative Anforderungsanalyse fuer die Praxis, 2007.

[SLS07] Andreas Spillner, Tilo Linz, and Hans Schaefer. *Software Testing Foundations : a study guide for the certified tester exam; foundation level - ISTQB compliant*. Rockynook, 2. ed. edition, 2007.

[TA08] Sylvie Trudel and Alain Abran. Improving quality of functional requirements by measuring their functional size. In *IWSM/Metrikon/Mensura '08: Proceedings of the International Conferences on Software Process and Product Measurement*, pages 287–301. Springer-Verlag, 2008.

[WRH97] W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt. Automated analysis of requirement specifications. In *Software Engineering, 1997., Proceedings of the 1997 (19th) International Conference on*, pages 161–171, May 1997.

Part III

Applications

Chapter 8

Control-Center Terminology and Application

Bratan Bratanov

Contents

Abstract: With the growing knowledge in software engineering area new approaches for controlling and improvement of software projects appear. Besides the general-purpose management tools, which do not solve essential state-of-the-art problems, concepts like ”software project control centers” are emerging. They aim at systematic quality assurance and improved management support having as background well-known paradigms from the theory - Goal-Question-Metric, Quality-Improvement-Paradigm - and integrating quality models like ISO 9126, McCALL, CMM. The tools themselves introduce new challenges in the software development domain due to their ambitious goals and complexity, which might be addressed only by a precise concept for architecture.
This paper presents an overview of the foundations and the different dimensions of control centers as well as a reference architecture for generic implementation. Furthermore some tools of this art already available in the industry are discussed and evaluated with regard to this architecture.

8.1 Introduction

The software development has evolved and improved significantly since the early 70’s, when the need for systematic engineering approaches was clearly identified, even though there still exist a lot of issues in terms of monitoring and controlling IT projects. According to several studies (e.g. the CHAOS report of the Standish Group [Gro08]) approximately 50% of the software projects end up over budget or schedule, despite the intensive use of process models and the increased know-how in project management. The reason for that resides in the lack of effective and efficient mechanisms and tools: to accurately identify the current project state; to identify critical situations like budget or time deviations; to determine weak points in the current project or product; or to estimate in generally whether the project goals would be achieved.
Software project control centers (SPCC) or ”software cockpits” might be defined as tools for process-accompanying interpretation and visualization of measurement data [JM02]. They could be figuratively referred to as an abstraction of control room (a term from the mechanical production domain), which is a central point for collection of all incoming data and visualization for various controlling purposes.
By this means they consist of methods to control software products providing the so called control indicators, rules and techniques for aggregation, combination and normalization of these indicators and rating levels for the measurements. Last but not least visualization techniques are essential part of the SPCC-concept.
SPCCs represent a major step in direction to solve the above issues. The main goals include the predictable software project execution as well as monitoring and evaluation of various quality characteristics of projects and products. Software cockpits aim therefore at improved controlling over project and intelligent support for decision making. Furthermore the SPCC helps improving processes and correspondingly products over time.

In order to fulfill these goals several tasks are performed, namely: collection, interpretation and visualization of data. The required information itself is gathered, processed and presented in context-, role-, and goal-oriented manner.
Important aspect of these tasks is the ability to perform them on-line meaning during project execution, since the product quality depends directly on the quality of the intermediate results on the one hand. On the other hand the measurement of projects makes sense exclusively during their execution except in the case when the still available information from past projects is intended to be reused as experience base.
The implementation of an SPCC though is rather challenging task due to the fact that it is an innovative area and therefore practical guidelines are deficient. A further problem is the inability to reuse approaches from other domains like general engineering or business management because of the specifics of software engineering.
Another challenge is the realization of customization, extensibility and reusability. But the primary problem remains in the projects themselves. Because often contradictory management goals or such with unclear interrelation are set. That causes difficulties in finding and deriving suitable metrics for controlling the project.
Typical solutions in the industry nowadays make use of standard project management tools like dashboards. They provide only a limited set of metrics and visualization capabilities. Also in general, they lack the possibility for project-specific or role-oriented customizations. Therefore they present only partial solutions for the above defined problems. More generic approaches are required, which could motivate more intensive application of SPCCs in the praxis.
In the following chapter are presented some of the theoretical foundations, which serve as basis for the implementation of software cockpits and facilitate the understanding of the SPCC concepts. Section 8.3 contains information about the different operational dimensions of SPCCs. Chapters 8.4 and 8.5 describe respectively a reference architecture for implementation of control centers and some frameworks from the industry compared according to it. In section 8.6 is presented a model for evaluation of SPCCs. Additionally some drawbacks and benefits of control centers in general are pointed out.

8.2 Background

Control Centers could be identified as a major domain for the application of software metrics. The successful implementation of SPCCs relies to a large extent on the systematical use of certain paradigms and approaches from the theory, which will be shortly discussed in this chapter.
Goal-Question-Metric (GQM) Since control centers are supposed to provide certain information in a goal-oriented manner, one of the primary questions is the choice of such key quality indicators, which will allow focusing on the stated goals for the concrete project. Quality indicators specify how measurements

are interpreted with regard to certain quality characteristics. There are several approaches for defining measurable project goals. The Goal-Question-Metric (GQM) [BCR94] is probably the most widely adopted in the praxis. That is due to its ability to specify concrete measures taking into account project specifics and organizational context.
The GQM paradigm represents a hierarchical model on three levels for mapping a set of metrics to a specific goal by additionally using control questions for facilitating the process of evaluating whether the goal is achieved.
On conceptual level (Goal) along with the goal an object (e.g. process, artifact) and view point (e.g. developer) are determined on which the goal is to be controlled.
The operational level is defined as set of questions refining the quality issue and breaking it down to measurable components. The final step is to define a metrics suite on quantitative level to measure the data in order to provide interpretable results for the quality estimation.
Having this break-down structure alone is not sufficient for solving issues like establishing relations between different quality aspects or comparison of the current project with past ones and deducing correlations. What is additionally needed is an abstraction or quality model, which serves as basis for the goal-oriented measurements. There are several widely used quality models, which could be used as foundation for building individual models, to provide the basis for a specific SPCC.
The tools discussed in this paper are based on the two product quality models **ISO-9126** and **McCALL** [OPR03].
ISO-9126 [ISO01] defines product quality based on internal and external quality attributes. The first ones address, how the product is developed e.g. size, tests, failure rate, while the second ones relate to how the product works in its environment. According to this model the product quality characteristics are divided into six major groups (functionality, reliability, usability, efficiency, maintainability, portability). They are furthermore refined on different level with subcharacteristics normally containing internal as well as external aspects. As advantage of this model is indicated exactly this classification of internal and external quality characteristics, even though ISO 9126 does not specify how these characteristics are to be measured.
The **McCALL** [JW77] model postulates a slightly different classification, namely: product operation, revision and transition. The product operations include characteristics like modifiability, reliability, usability, etc., which help to estimate whether a product could be quickly understood, efficiently operated and is capable of providing the desired results. The product revision handles the ability of a product for error correction and adaptation (e.g. maintainability, flexibility). The product transition deals with characteristics like portability and reusability. The McCALL model in contrast to ISO 9126 also specifies metrics for the proposed characteristics. This could also be pointed out as disadvantage since some of them are claimed to be subjective.
An important aspect of SPCC is the evaluation of the process quality especially the ability for improvements. The quality improvement paradigm - **QIP** [Bas93] offers such quality-oriented improvement approach. The key concept of this

approach is based on choosing a process model according to the specified project and organizational goals instead of vice versa. The QIP consists of six consecutive phases: characterize, set goals, choose process, execute analyze and package. During the different phases the environment of the project is characterized and quantifiable strategic organization- and specific project-goals are defined. Afterwards a process is chosen based on the above facts and executed providing feedback for the goal achievement. Additionally the gathered information is analyzed for evaluation and recommendations concerning the applied practices and processes. Finally the gained experience is consolidated and persisted in an experience base for future projects.

8.3 Different dimensions of control centers

One way for breaking down the complexity of SPCC-implementation is to understand the different dimensions, in which they operate. A successful implementation must be able to systematically integrate several orthogonal dimensions. Below a selection of the dimensions presented in [JM02] is briefly discussed.

8.3.1 Usage purposes

The possible usage purposes of an SPCC influence its functionalities on a large scale. Thus a generic solution should take various organization- and project-specific purposes into account. Also it should be able to extend its functionality according to newly defined purposes. Most common examples of such purposes are:

Monitoring: observes the current project state in terms of product, process and resource performance

Assessment: processes monitoring information and forms judgments of project quality characteristics

Analysis: identifies quality and goal deviations by examining the context and identifies the possible causes

Comparison: uses predefined data from finished projects or guidelines in order to estimate the current state

Prediction: applies techniques to predict or simulate future project behavior based on current state and mathematical models

Improvement: proposes corrective actions based on identified shortcomings or deviations and considering the project goals

8.3.2 Main tasks

The main tasks represent a decomposition of the SPCC workflow, which provides the value to the users. The term ”workflow” is defined as the sequence of activities performed during the integration and the operation of SPCCs. The first step in this workflow is **building/implementing a quality model**. The model is then used to compare the current project with past ones or with guidelines. Well-known quality models like ISO 9126 and McCALL are recommended as basis. Additionally the individual model could integrate process models helping to synchronize project activities with the control center functions (e.g. data collection with development tasks). SPCC is a goal-oriented

Table 8.1: Enhanced GQM model

Goal	Question	Subquestion	Metric
Purpose: Evaluate Issue: The easiness of identifying styles, structure, behavior and parts of maintenance	Q3700 Are the functions not too complicated?	Q3701 Is the function-call nesting not deep?	MFn095 Depth of layers in call graph
Object: source code Viewpoint: developer		Q3702 Is the logic not too complex?	MFn072 Cyclomatic number MFn069 Estimated no. of static paths

environment, therefore **goal-definition** is required. The proposed approaches use the GQM paradigm for this purpose. One optimization of the standard GQM-model with respect to SPCC is the introduction of additional hierarchical levels [WNF+07]. As 8.1 shows, the questions might be defined at high abstraction level and refined on the next level with concrete e.g. implementation-specific sub-questions. These could be specific with respect to the controlled project, the used programming language and etc., thus they are more easily mapped to metrics. This refinement improves the reusability of SPCC by defining common parts (commonality points) and modification areas (variability points). The approach involves also the next important task **goal-oriented derivation of measures**.

Once the goals and the metrics are determined the next tasks could be performed. These are as follows: **data collection, data aggregation** and **processing** and **data visualization**. Collecting data is a continuous activity during the whole life-cycle of the project. It involves interaction with various heterogeneous sources like project plans, source code, tests, and etc.

The gathered data often need to be aggregated and normalized in a way to suite the intended purposes. Finally it must be represented in an understandable way for the involved stakeholders.

8.3.3 Scopes and granularity

This dimension addresses the integration specifics expected from a SPCC in order to be used on different levels. The following three major scopes are distinguished: project-specific, past project and future project scope. On a project-specific level the variation points must be adjusted accordingly e.g. to the applied technologies, processes. The proposed GQM enhancement allows not only reuse of certain parts for different projects, but also provides the opportunity to define a granularity in a single project by applying only the relevant sub-questions and metrics on smaller components (e.g. product modules, project milestones), which is essential for the different roles-oriented visualizations. With respect to this scope an SPCC provides generally real-time feedback and improved control over current projects.
The past-project scope deals mainly with packaging and reusing experience from previous projects through the design of building experience bases.
The future project scope on the other hand aims at improvement strategies relying on paradigms like QIP.

8.3.4 Roles

The roles dimension describes the potential users of an SPCC. It addresses not only the output information that these users need, but also the input data that they should provide for the proper functioning of the SPCC.
A **project manger** for example could be supported through SPCC by providing him with overall information about the current project state together with additional informations. These could include identified quality deviations, recommendations for corrective actions, simulation of alternative plan executions. As input a project plan, project goals and etc. are expected.
Another role that could benefit from an SPCC is the **quality manager**. The control center could provide information about the current or predicted goal violations or to identify best practice experiences. The role itself could influence the operation of the SPCC by providing quality models and measurement plans.
On fine-granular level a **developer** could be supplied with information regarding his components e.g. violations of specifications, guidelines, schedule.
External stakeholders or customers might receive highlight information about qualities of the product or estimations for the project. In an agile environment the customer could be actively involved by providing input information like goals, change requests and so forth.

8.4 The reference architecture

A concrete software cockpit implementation should consider all the dimensions, described in the previous chapter. Therefore any architecture for generic control center applications should focus features like extensibility, customizability and reusability. In particular two architecture aspects important for the successful SPCC implementation should be considered: logical and physical.

8.4.1 Logical Architecture

The presented conceptual logical architecture On figure 8.1 is adopted with small modifications from the paper [HM08]. The reason for that is that it focuses to a large extent on the above features and claims to handle in systematically the different dimensions. The other researched approaches offer partial solutions excluding certain aspects not relevant for their stated goals. This fact prevents them from being reasonably applied as reference for the comparison of existing tools.
The chosen approach represents classical layer architecture with three distinguished layers.

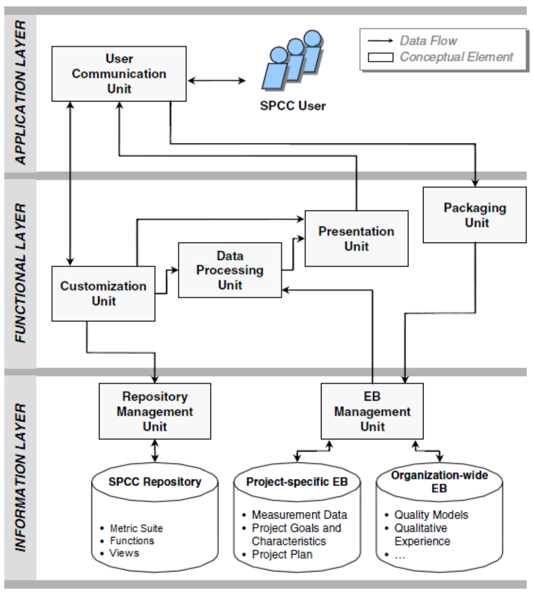


Figure 8.1: Logical reference architecture of software project control center (Source: [HM08])

Information layer

The information layer is basically responsible for collecting and persisting the information necessary for the SPCC functionality. It consists of two major components dealing with current measurement data, experiences from past projects and organization-strategic information. The repository unit encapsulates generic reusable components like the metric suite or even higher-level techniques for serving to a purpose from the purpose dimensions referenced bellow as functions. Furthermore it contains views, which specify measurement results of the same purpose for different roles or also from the viewpoint of the three major groups of measurement entities - products, processes and resources.
The other major component in the information layer - the experience base - refers to the storage of the measured data and the relevant context information. It is divided into a project-specific and an organization-wide unit.

Functional layer

The functional layer relies on the services provided from the information layer for deriving useful information in a concrete context. It contains a customization unit, which initializes and tailors the repository components according to the project goals.
Relying on this customization a data processing unit executes the aggregated metrics and analyses the results. During this operation the relations between different metrics and functions are analyzed and rating levels are assigned, so that each metric might be weighted.
The results of the comparison in this module are afterwards delivered to the presentation unit. It initializes the tailored views and integrates the required data received from the processing unit.

Application layer

The last layer takes care of the viewpoint-oriented visualization of the results. It contains authorization mechanisms allowing the SPCC to deliver specific functionalities to different users. Such functionalities might be: role-oriented visualization, forms for providing information, tailoring functionalities - based on the user's role.

8.4.2 Physical components

Apart from the logical components physical details and certain specifics for integration in development environments should be considered. As mentioned earlier many solutions nowadays rely on dashboards mainly based on spreadsheets (e.g. CAST's AD Governance Dashboard, Pentaho, Jaspersoft). For more generic solutions a database handler is a necessity especially when considering distributed environments. The critical point on physical level and implementation-oriented design is the heterogeneous nature of an SPCC. Heterogeneous in terms of the data sources used as input, but also in terms of controlling distributed development. An SPCC must normally provide connectors to a huge variety of external systems like effort tracking-, bug tracking- and other general-purpose management-systems. Because they already contain most of the required information. Analogously different analysis engines and output formats should be integrated.
Handling distributed development is the other key factor of the physical architecture. In this case an integration and synchronization of the data from the different locations is needed in order to estimate the overall project state. It also brings additional requirements for the granularity of the measurement results e.g. controlling per location.
In the next chapter are presented three different solutions from the industry and are pointed out some strengths and weaknesses with regard to the architecture proposed above.

8.5 Industrial tools and frameworks

Most of the software cockpits in the industry offer partial solutions according to the architecture described in the previous section. Generally they lack either role- or purpose-oriented customization or support only a fixed number of metrics. In the following chapter three different solutions are presented and shortly evaluated according to their usefulness, customizability and improvement potential. All three of them address most of the aspects covered in the reference architecture. In particular the physical aspects like integration with existing tools and distributed development are thoroughly satisfied.

8.5.1 WebME

WebME [Tes97] is a web-based tool developed by NASA, whose primary target group is project managers. The architecture is similar to the reference one from 8.1. Despite the fact that it is meant for project managers, it turns out to be highly customizable for different roles and purposes due to the embedded scripting language. WebME focuses on distributed development, whereby

heterogeneous data sources from different locations could be easily integrated using the script language. The framework allows real-time measurement and feedback. Since it is web-based it is platform independent and could be accessed from any location without the need of additional configuration.
Its main deficits are the fixed set of functions and metrics. The underlying model is also predefined and could not be extended. This drawback is compensated with the fact that a sophisticated predictive model is applied. It relies on the predictive CMM and economic forecasting models. It claims to dynamically evolve using only past projects of the same shape when estimating the current one. Its success in different domains is though not proven. The ease of use seems to be challenged because of the use of a custom designed not very intuitive script language instead of applying a well-known one.

8.5.2 Specula

Specula [HM08] is a J2EE-based framework developed from the authors of the proposed reference architecture. Therefore it is fully compliant to it even though simplified in some aspects. The main focus lays in the extensibility and reusability with respect to functionality and customizability in terms of application contexts. The platform is web-based as well and by this means platform-independent. It supports distributed development as the previous one. The framework also allows defining heterogeneous sources either automatically - by providing java connectors - or by manually inserting information through forms. An advantage compared to WebME is the ability to add metrics and extend the quality model. The case studies indicate a positive result with respect to early detection of risks and deviations.
The drawbacks of the approach are the difficult integration into the software development cycle and the tendency to include too much data in certain views. The last aspect causes scalability problems and has negative impact on the ease of use. The framework also does not offer an automated derivation of views from GQM plans. The absence of such automation enforce manual intervention, which in turn requires deep understanding of the underlying architecture and increases efforts. Automation is a crucial factor for user acceptance. The weakness here could be justified by the fact that Specula is a research prototype.

8.5.3 SQO-OSS

Software Quality Observatory for Open Source Projects (SQO-OSS) [GKS$^+$07] is a plugin-based service-oriented framework for evaluation of open-source products and processes. The project contains also a sample implementation with several plugins and integrated data from a few examined projects. Due to this concept of plugable architecture, which encapsulates most of the com-

ponents are plugins able to be added or removed on demand, the platform is extremely extensible and customizable.
The nature of open source projects - the primary domain of this framework - demands from such products on a large scale interoperability and simplicity by integration of external systems. The reason is that most of the considered projects are distributed and use heterogeneous sources. SQO-OSS achieves these goals first of all by applying this pluggable architecture. Additionally it uses well-known techniques for integration of components - xml scripts and web services, thus avoiding thorough understanding of complex APIs. To the benefits of this platform counts also the mere fact that it is open source. Therefore allowing everyone to contribute - meaning that huge amount of plugins could be available, when widely adopted. The numerous open source products on which it could be applied, deliver also rich experience base for comparison. This could cause on the other hand big overhead for configuration, when applying it on trivial projects. In this case extensive filtering of data would be needed, so that only ”alike” projects are explored and compared. As deficit of the framework could also be pointed out the absence of explicit concept for goal- and especially role-oriented processing and presentation. In contrast to the reference architecture, this one does not distinguish between layers. The separation of concerns is therefore delegated to the plugins themselves.

8.6 Discusion

In this chapter a model for evaluation of software cockpits on a high level is presented. Furthermore some general threads and opportunities for SPCCs according to this model and with regard to the researched SPCC implementation approaches are mentioned.
Evaluation
A fundamental question when delivering a software product is how well does it serve its purposes and would it therefore be accepted by the users. As seen by the description of the backgrounds and the reference architecture, the implementation of such solutions is complex and contains many pitfalls. This combined with the fact that SPCCs are relatively new area thus not many best practices are established, encourages both providers and customers to have a good starting point for evaluation of software cockpits.
The Technology Acceptance Model (TAM) [CHM$^+$07] suites perfectly for this purpose. TAM is an information system theory modeling the degree to which users accept and use a technology. In SPCC context three features are comprehended at abstract level:

ease of use as the degree to which a SPCC could be integrated into the development environment and used free of effort

usefulness as the degree to which a system enhances job performance or bring benefits for the user and in particular supports effective project control

improvement potential as the information provided in order to make transparent decisions and apply improvement strategies

Empirical studies performed on top of these characteristics establish for software cockpits a trend for a reverse correlation between ease of use and usefulness. In end effect more holistic solutions require additional effort during the initial configuration in particular context, but on the other hand facilitate to a greater extent project activities. This tradeoff justifies the fact that many tools in the industry simplify or reduce the dimensions and functionalities mentioned in the previous sections. A holistic approach does not pay off for small- and medium-size projects. For big projects or organizations with many "alike" projects though this tailoring represents a small part of the overall effort, thus making generic solutions preferable.
Nevertheless even partial solutions show big improvement potential, when applied methodologically and with clear goals. In the following sections several common threats and opportunities for SPCCs based on empirical results from industry [CHM+08] are presented.
Drawbacks
As mentioned above integrating an SPCC in a development environment might be time-consuming activity. Considering the fact that different views for different roles have to be integrated into a single concept demanding inter-view communication, since the different stakeholders focus on different aspects and prefer different interfaces. Causing such interface inhomogeneity complicates the definition of multidimensional goals and might reflect in exuberant data in the separate views. This on its behalf reduces the understandability and scalability of the presented information.
Another group of problems resides in the specifics of the adopting organization. Often the definition of common strategies and goals in particular in big organization is a tough task. The absence of consensus about common organization-wide controlling results in application of the SPCC on single-project level eventually significantly reducing its improvement potential.
On technical level the main challenges for software cockpits are connected with the extent to which automation is utilized. On one hand many tools already deployed do not provide public APIs for interaction and by this means introduce redundancy and potential inconsistencies. On the other hand validation and filtering of information might turn into serous obstacle influencing the accuracy of results.
Accuracy also is dependant on estimating the importance of the different quality indicators. A lot of the indicators are well-known in the research area, but there is still little knowledge how they affect quality in industrial environment.
Benefits
An important difference between the existing partial solutions like dashboards and SPCCs is that the first ones offer a fixed company-specific functionality without clean methodological support. They allow therefore no possibility for

customization and tailoring, when the goals are changed. Despite the above deficits dashboards are still as time-consuming as the application of holistic tool.
SPCCs on the contrary offer a more clear control over projects by accurate quantitative presentation of the current project state. The concept of a control center as central information control unit simplifies the understanding of multidimensional goals. An SPCC provides consistent information to all the involved parties and helps with early detection of plan deviations, goal violations and project risks. From organizational point of view SPCCs gives the opportunity for standardized controlling over multiple projects, easy access to data from different contexts and facilitates the management of distributed projects.
Future directions
Applying Software project control centers in the industry is a hot topic, which might turn into a successful solution for many existing problems, the software companies face nowadays. An interesting question is whether the underlying concepts could be adapted to other domains as well. Such integration could improve the collaboration and reduce the gape between application domains and the corresponding software for them - one of the initial problems of software engineering. Such adaptation would be a significant improvement of the processes in inter-disciplinary organizations, which currently have to support a complex set of disjunctive controlling mechanisms.

8.7 Conclusion

The concept of SPCCs has emerged from the necessity of improved project control by using relevant quantitative information for the appropriate estimation of the current project state. The use of metrics though turns out to be a challenging task. It requires good understanding of the underlying processes, well-defined and not contradicting project goals and systematic approach when mapping metrics to the concrete goals.
A key factor when adopting and implementing software cockpits is that they should be generic enough be applied in different context. The integration in concrete development environments demands certain dynamics with respect to applying measures and indicators, collecting and visualizing data. From this point of view strict requirements on extensibility, customizability and reusability are yielded.

Bibliography

[Bas93] Victor R. Basili. The experience factory and its relationship to other improvement paradigms. In *ESEC '93: Proceedings of the 4th Eu-*

ropean Software Engineering Conference on Software Engineering, pages 68–83, London, UK, 1993. Springer-Verlag.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. 1994.

[CHM+07] M. Ciolkowski, J. Heidrich, J. Munch, F. Simon, and M. Radicke. Evaluating software project control centers in industrial environments. In *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, pages 314–323, Washington, DC, USA, 2007. IEEE Computer Society.

[CHM+08] M. Ciolkowski, J. Heidrich, J. Munch, F. Simon, and M. Radicke. Empirical results from using custom-made software project control centers in industrial environments. pages 243–252, New York, NY, USA, 2008. ACM.

[GKS+07] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, and D. Spinellis. Software quality assessment of open source software. volume A, pages 303–315, Athens, May 2007. New Technologies Publications.

[Gro08] Standish Group. Chaos summary 2008. 2008.

[HM08] Jens Heidrich and Jürgen Münch. Implementing software project control centers: An architectural view. In *Proceedings of the International Conferences on Software Process and Product Measurement*, pages 302–315, Berlin, Heidelberg, 2008. Springer-Verlag.

[ISO01] ISO/IEC. Iso/iec 9126: Information technology - sowtware product evaluation. *Software Engineering*, 2001.

[JM02] Jens Heidrich Jürgen Münch. Software project control centers: concepts and approaches. *Software Quality Control*, 2002.

[JW77] P.K. J.A., Richards and Walters. Factors in software quality. *National Technical Information Service*, I-III, 1977.

[OPR03] Maryoly Ortega, María Pérez, and Teresita Rojas. Construction of a systemic quality model for evaluating a software product. *Software Quality Control*, 11(3):219–242, 2003.

[Tes97] Zelkowitz M.V. Tesoriero, R. The web measurement environment (webme). *In: Proceedings of the 22nd Annual Software Engineering Workshop (SEW)*, 39(3):281–289, 1997.

[WNF+07] Hironori Washizaki, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada, and Hiroyuki Watanabe. A framework for measuring and evaluating program source code quality. In *PROFES '07: Proceedings of the 8th international conference on Product-Focused Software Process Improvement*, pages 284–299, Berlin, Heidelberg, 2007. Springer-Verlag.

Chapter 9

Source Code Metric Tools

Stefan Vesselinov Cholakov

Contents

Abstract: This paper starts by motivating the usage of source code metrics and states their main applications. A classification of source code metrics is offered and some of the most common ones are explained. The issues with the definitions of metrics are pointed out and backed up with examples. Some of the popular commercial and open source tools on the market are presented, focusing mainly on their characteristic features, the supported metrics as well as their calculation and interpretation when applicable.

9.1 Source Code Metrics

9.1.1 Motivation

A metric is a standard of measurement. It maps an aspect of the object, on which it is applied, to a measurable quantity, such as a number. Metrics are very important, since, as motivated by Tom DeMarco,

"You cannot control what you cannot measure."

As quantitative methods have proved very helpful in other areas of science, computer scientists have also been trying to bring similar approaches to software development.

Every developer or project manager has certain goals and objectives. In order to determine if a goal is reached or what has to be done to reach the goal, a measuring system needs to be used. Even when not explicitly defined, a measuring system always exists. Source code metrics are intended to be an explicit measurement system that is as objective as possible. Once the measuring system is established, data can be collected, the measurements can be evaluated and compared to the benchmark thus obtaining the benefit of being able to evaluate progress, effectiveness, quality, etc in a quick and reliable manner.

In order to provide an intuition about the multiple benefits of source code metrics, consider Lines of Code as an example. One could define it as the number of lines of code written by each programmer within a week. Further on, a metric that counts the number of detected bugs per thousand lines of code could be defined. By comparing each programmer's metric against the benchmark of "fewer than x bugs per thousand lines of code", various benefits con be extracted. Such "derived" metrics could be used to, for instance, describe (quantify) the quality of work of the programmers in a company.

In the following, a broader look into the various applications of metrics is offered. The main applications of source code metrics tools are classified. The section tries to summarize the benefits earned through the usage of source code metrics tools in the different areas of their application.

9.1.2 Applications of source code metrics

One of the main goals of source code metrics tools is certainly to provide developers and managers with a **better insight into the code**. Since the quality and the architecture of a system deteriorate with time, it is important to know whether it still conforms to the initial requirements.

Refactoring is a major application area for source code metrics. Code metrics can help to make the decision on **when to refactor** a system and further to understand at which points should be reworked and what the nature of the refactoring should be. Identifying structures where refactoring should be applied often is explained with subjective perceptions like ”bad taste” or ”bad smell”. But in order to allow refactoring, the management normally needs a proof that this is needed at all and that refactoring would increase the system's business value. Metrics can be used as an **objective measure of the quality** of a system. Metrics can support these subjective perceptions and thus can be used as effective and efficient way to get support for the decision **where to apply which refactoring**.

By taking advantage of code metrics, developers can understand which types and/or methods should be tested more extensively. Development teams can identify potential risks and **track progress** during software development.

Understand the current state of a project is essential before making a decision to **reenginneer** a system. One needs to know the strengths and weaknesses of the existing system in order to decide which parts of the existing system should be kept, modified or thrown away.

Code metrics could also help programmers to write code that adheres to a **coding standard**.

9.1.3 Classification of source code metrics

Source code metrics could be classified in various ways. Here standard established metrics are classified into three groups - Complexity, Software Architecture and Structure and software Design and Coding, according to [iSA09] (See Table 9.1). Some of the most common ones are discussed in greater detail thereafter.

As found by [ea08], complexity metrics have a good performance in distinguishing between fault-prone and not fault-prone classes. Those can be divided into the size, interface complexity and structural complexity metrics. An example for a size metric is **Lines of code (LOC)** - perhaps the most widely known source code metric. Though it looks clear what it measures, there are multiple implementations of this metric that are discussed in section 9.1.4.

Higher values indicate higher overall complexity.

Cyclomatic Complexity (CC) is an example for a structural complexity source code metric. CC was developed by Thomas McCabe and counts the number of decision paths in a given method. Variants for its calculation are available, though higher cyclomatic complexity typically implies more possible points of failure and therefore a higher number of required test scenarios. Some sources consider methods with CC above 15 as complex and hard to maintain.

Weighted Methods per Class (WMC) measures the sum of the methods of a class, weighted by a certain factor, that differs according to implementation. Please refer to the next section for more information on the variations of its calculation.

Table 9.1: A classification of source code metrics

Metrics Group	**Subgroup**	**Common Metrics**
Complexity		
	Size	Lines of Code (LOC), Code coverage
	Interface Complexity	Number of Attributes, Number of static Attributes, Number of Methods(NOM), Number of parameters
	Structural Complexity	McCabe Cyclomatic Complexity (CC), Weighted Method Count/Weighted methods per Class (WMC), Response For a Class (RFC)
Architecture and Structure		
	Inheritance	Depth of Inheritance Tree (DIT), Number Of Children (NOC)
	Coupling	Afferent Coupling (Ca), Coupling Between Objects (CBO), Change Dependency Between Classes (CDBC), Change Dependency Of Classes (CDOC), Efferent Coupling (Ce), Coupling Factor (CF), Data Abstraction Coupling (DAC), Instability (I), Locality of Data (LD), Message Passing Coupling (MPC), Package Data Abstraction Coupling (PDAC)
	Cohesion	Lack of Cohesion in Methods (LCOM), Tight Class Cohesion (TCC)
Design and Coding		
	Documentation	Lack Of Documentation, Documentation Density
	Code Conventions	

The group ”Architecture and Structure” contains package metrics and dependencies and comprises of inheritance, coupling and cohesion source code metrics.

Depth of Inheritance or Depth of Inheritance Tree (DIT) is a very common structure source code metric. It is defined as the maximum inheritance path from a class to the root class. On the one side, inheritance is a fundamental concept in object-oriented programming. On the other, the deeper a class is situated in the inheritance tree, the harder it becomes to maintain and test it. Typically, a class that inherits directly from the base type Object has a DIT value of 1. In the literature, different acceptable ranges for DIT are defined that have an upper bound between 4 and 8. Microsoft suggest a maximum of 5.

Also common are Robert Martin's code metrics - Afferent and Efferent Couplings, Instability, Abstractness and Distance.

Afferent Couplings (Ca) refers to the number of classes directly depending on the analyzed one. Classes with high Ca value are considered to have many responsibilities. Ca can be applied on other levels as well, for example on package level. Ca value of 0 indicates that a class is not used.

Efferent Couplings (Ce) refers to the number of classes, on which the analyzed class depends. High Ce value of a class indicates that it is highly dependent. Ca and Ce might be very helpful when compared to the size of a class.

Instability (I) is calculated as the ratio I = Ce / (Ca + Ce) and indicates the level of changeability of a class or a package. The values of I are in the range between 0 and 1, where 0 indicates a stable class and 1 an unstable one. The more a class depends on other classes, the more unstable it is. If many classes depend on the analyzed one, modifications of that class are less likely to be required when making changes somewhere else in the system, therefore the class is stable. GUI-related packages are recommended to have I values close to 1, while data access-related classes – close to 0.

Abstractness (A) is the ratio of the number of abstract to the number of concrete classes/ modules in a package. A value of zero indicates a completely concrete package and a value of one - a completely abstract. Alternatively, some tools such as the plug-in for Eclipse "Metrics", calculate A as the ratio the sum of abstract classes and interfaces divided by the total number of classes in a package.

In the next section the problems with metrics definitions are addressed and some of the implementations of this metrics are discussed.

9.1.4 Issues with the definitions of metrics

There are several issues with the definitions of source code metrics. The developers of a metric not always clearly define it, sometimes its calculation is left open as an implementation decision. This is the case with WMC. WMC is defined as the weighted methods per class. The weight is left as an implementation decision. Often the weight for all methods is set to one. In this case WMC is equal to the number of methods in a class. Alternatively, the cyclomatic complexity of a method can be used as its weight, resulting in a WMC value equal to the cyclomatic complexity of the class.

Another issue with the definitions of metrics is that some well-established ones are calculated differently by the different source code metrics tools. For example, Lines of code(LOC) is calculated by some tools as the total number of lines of a project, while others take only the non-empty lines. Still others additionally exclude the non-empty that contain comments. LOC could be defined as the sum of the executable lines of code of all methods in a class. This further excludes the import/using statements, as well as the method signatures. It is also not uncommon to calculate LOC as only the lines, which contain a semicolon. Alternatively, LOC could be measured based on intermediate language (Microsoft IL, Java bytecode), instead of source code.

The example above also demonstrates a further issue - though there are different implementations for the calculation of a metric, sometimes they are all called with the same name.

Additionally, the interpretation and usage of a metric could deviate from the original intention of its developer. This poses the question whether the results of such metrics are still as exact as expected. Due to all the issues mentioned here, one should consider the implementation and the interpretation of a metric.

The next section defines and point out the advantages of source code metrics tools. Commercial and non-commercial tools that were found interesting by the author are shortly presented. The features that differentiate them from other products on the market are discussed. As already mentioned, the tools often measure different metrics in different ways and interpret the results differently. The metrics, their calculation and interpretation are addressed where applicable.

9.2 Tools

Applying metrics manually would be a a tedious, expensive and error-prone task, which is not practicable. Additionally, there is a high probability that those results would not objective. Tools on the other hand, are capable of delivering automatically calculated measurement in a very quick manner typically only by a click of a button. Metrics tools are the only way to deliver objective answers [Fle09].

Tools, offering features beyond calculating the common basic metrics, could help developers, managers and analysts identify risks and detect symptoms in the source code that possibly indicate a deeper problem. An example of a so called ”bad smell” is duplicate code – identical or similar code segments found in more than one location. Duplicate code implies high coupling between those locations and thus lower maintainability of the system. Some metrics tools detect such design compromises and are very useful when performing refactoring.

Since developers look at small portions of code through the IDE window, it is possible that they loose the overview of the system. This would inevitably result in reduction of the code quality. Advanced source code metrics tools offer additional views on the system that are helpful in retaining the overview of the system. Such views are, for example, dependency graphs, charts, showing the methods with too high cyclomatic complexity, etc.

There is a wide variety of source code metrics tools available on the market. This section addresses some of the most widely used tools from different perspectives, such as the measured metrics, reporting capabilities, configurability, etc. The scope of this paper are source code metrics tools are capable of analyzing **Java or C#/.NET** code. For those that additionally support other languages, the supported ones are pointed out.

The tools presented here were divided into commercial and open source. **The choice of the particular tools** was made primarily based on a search on the Internet. Factors for the choice of tools were the supported programming languages, the range of supported source code metrics and the additional advanced features provided. Further on, the customers of the tools and the tools’ degree of acceptance in the industry were considered. Users’ feedback, tool usability and documentation were also taken into account.

9.2.1 Commercial Tools

SonarJ

SonarJ is a powerful source code analysis tool for Java systems. It is delivered either as a stand-alone application or as an Eclipse plug-in. SonarJ supports a wide variety of source code metrics, divided into groups that can be seen on the left pane of Figure 9.1. A metric can be calculated within different scopes – system, subsystem, package, directory, etc. Each metric has its own set of scopes. In the histogram view on the right pane of the figure, the number of packages having a certain Instability index are displayed.

One could also set thresholds for the metrics. If the threshold for cyclomatic complexity is for instance set to 15, every developer that tries to write code with higher complexity will get a warning. Additionally, tolerance ranges in

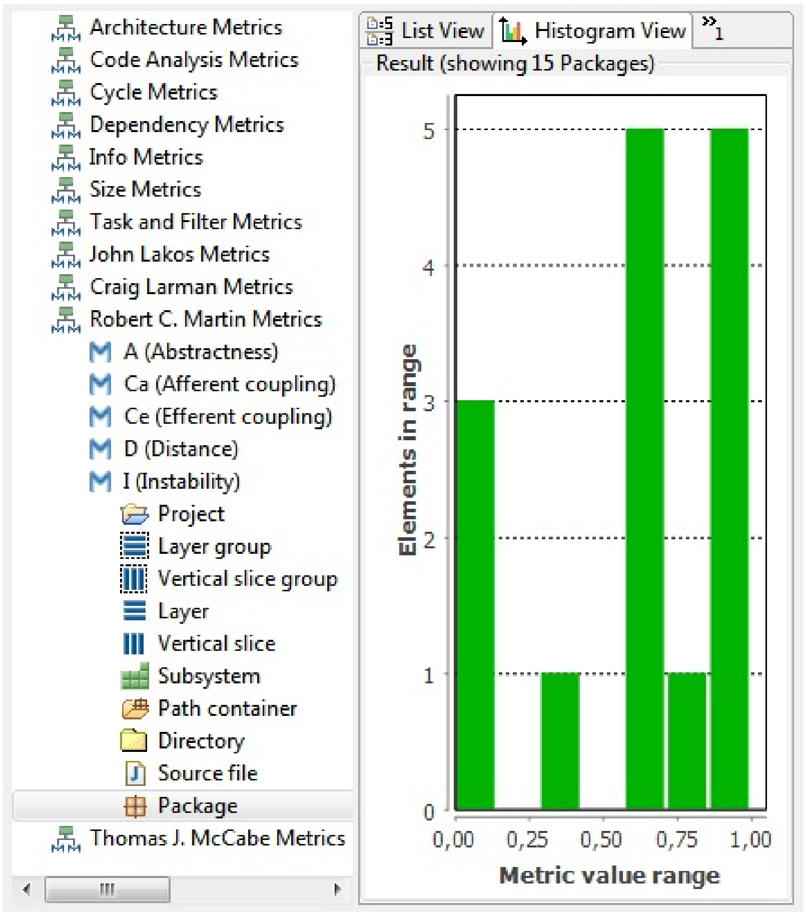


Figure 9.1: SonarJ - Metrics Tab

percent can be set for the metrics. Elements (classes, packages, etc.) become ”suspicios” if they fall within those ranges.

In the Exploration tab, dependencies among elements are visualized. Figure 9.2 displays the package structure of a system. By expanding and collapsing the branches of the tree on the left, one could drill-down to a class level. In the figure the ”gui” package is selected. The green arches show that it depends on the controller and model packages, among others. An arch on the right would indicate a cyclic dependency. The cycle viewer is an additional help for identifying elements for refactoring.

SonarJ offers the possibility to plan refactoring and simulate refactorings without actually applying them. For further information, please refer to the official documentation at [hel10].

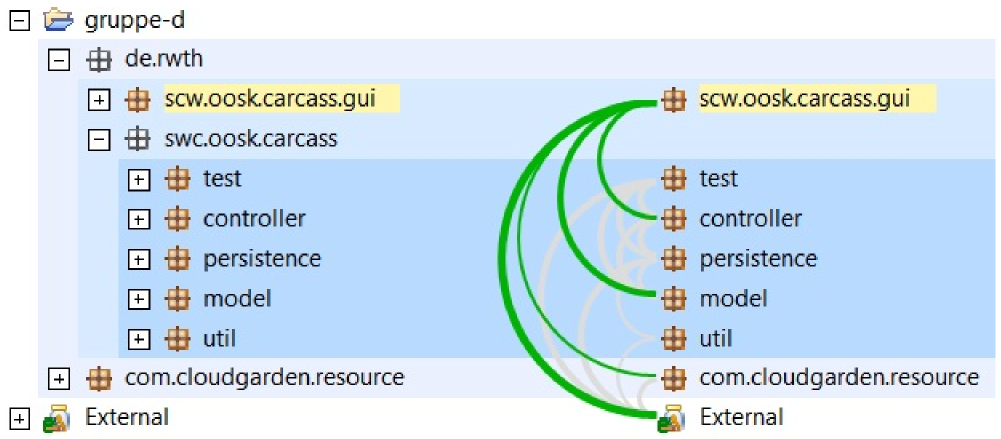


Figure 9.2: SonarJ's graph "Logical Structure of a System"

Microsoft Code Metrics

Microsoft Code Metrics is integrated in Visual Studio Team System - Team Suite 2008. It supports five metrics, which are displayed as columns in a table. Lines of Code excludes white space, comments, braces and the declarations of members, types and namespaces. Microsoft Code Metrics measure Depth of Inheritance in the standard manner. The Class Coupling metric indicates the total number of dependencies that the item has on other types. This number excludes primitive and built-in types such as Int32, String and Object [Cor07]. Low Class Coupling can indicate candidates for possible reuse.

Microsoft Code Metrics calculate Cyclomatic Complexity by counting the number of decision points (such as if blocks, switch cases, and do, while, foreach and for loops, &&) and adding 1 [Cor07].

Maintainability Index is an number between 0 and 100 indicating the overall maintainability of the member or type. At the namespace and assembly level, this is an average of the maintainability index of all types contained within it. Maintainability Index is a derived metric, calculated as a function of Cyclomatic Complexity, Lines of Code, Halstead's Volume metric and others not explicitly mentioned by Microsoft. The latter measures the information content of an algorithm implementation in bits, based on the number of unique and total operators and operands in the analyzed segment. A Maintainability Index below 9 indicates code that is complex and hard to maintain. The range for Moderate Maintainability is between 10 and 19 inclusive, and above 20 the maintainability is considered as "High".

RSM (Resource Standard Metrics)

Resource Standard Metrics is a source code metrics and quality analysis tool for ANSI C, ANSI C++, C# and Java for use on all Windows and UNIX operating systems. According to the list of companies using it, it is a widely accepted and used tool. Some of RSM's customers in Germany include Alcatel, Commerzbank, Blaupunkt, Daimler Crysler Aerospace, Deutsche Telekom, HP, Siemens, Sony and others [LLC09]. RSM is a command line program that enables scripting and embedding into programming IDEs. RSM Wizard is free GUI which runs on top of Resource Standard Metrics. The tools supports various operating systems and boasts to be the fastest tool for measurement of code quality and metrics. Due to the evaluation limitations the author has not verified this.

The tool supports, according to the documentation, around 100 metrics, which are split into scope groups: function, class, namespace/package, file, project and baseline metric differentials. Three metrics for the measurement of lines of code are defined per scope - total, effective and logical statements Lines of Code. That results in 24 metrics that are variants of Lines of Code. Other supported metrics are blank lines, comment lines, function points (a derived metric, based on lines of code), number of input parameters, number of public, private, and protected data attributes and methods, cyclomatic and interface complexity. For a complete list of supported metrics, please refer to the official documentation [LLC09].

Reports are generated for the selected metrics in textual, CVS, XML or HTML format. In the author's opinion, the reports are not clearly laid out, contain redundant information and do not suggest the interpretation of the results. The supported metrics do not promote improved clarity into the system under measurement.

Analyst4j

Analyst4j is a source code metrics tool for Java. It is distributed as a standalone application and as an Eclipse plug-in. It offers a variety of source code metrics on method, class, file and package levels. The metrics, their calculation and interpretation are well-described in the official documentation at [Ana09].

The results of the metrics are visualized in graphs that are, in the author's opinion, more comprehensible than a list of numbers. Thresholds for the metrics can be set as well. Metrics can be mapped against each other with the help of the comparison analysis feature. For example, Weighted Method Complexity versus Inheritance Depth of a class can be displayed in one chart to reveal complex classes with lots of functionality that might be candidates for refactoring.

A nice feature offered by the tool is metrics search. One could select the desired metrics, set thresholds to them and connect them by means of logical operators. The queries created in this manner can be saved for later evaluation. One could use this feature to prioritize coding and refactoring activities.

Analyst4j supports the detection of anti-patterns. On the product web page [Ana09] the detected anti-patterns, such as ”Spaghetti code”, ”Blob classes” and ”Swiss knife classes” are well described together with the symptoms of those anti-patterns in terms of ranges for source code metrics.

Other commercial tools

Krakatau Suite Krakatau Suite [Sof09] offers three metrics tools - Krakatau Professional, Project Manager and Essential Metrics. It supports C/C++ and Java. Krakatau Professional offers over 70 source code metrics including Halstead size metrics, complexity and object-oriented metrics. The results of the metrics are visualized in spider graphs, histograms and tables. Tolerance ranges can be set for the metrics.

Krakatau Metrics PM can compare two project releases and display changed, deleted and added source lines of code between the projects. This feature could be used to infer the developer activity and detect problem areas in source code. Essential Metrics is a command line metrics tool, which makes it suitable for inclusion into a build process.

Further source code metrics tools are **NDepend** [S.A09] (supports .NET languages), **XDepend**(for Java) and **Semantic Designs Java/C# Metric tool**. Note that NDepend and XDepend have little in common with the similar sounding open source tool JDepend.

9.2.2 Open Source Tools

XRadar

XRadar is an open extensible code report tool, the architecture of which is based on java, xml and xsl [XRa09]. Currently it supports only Java. XRadar uses other open source products internally, such as JDepend, JavaNCSS, PMD and Checkstyle, some of which are shortly referred to in the section 9.2.2. It covers the standard range of source code metrics, such as complexity, architecture and structure metrics. XRadar also checks for class design errors (such as visibility of attributes), duplicate code and code layout.

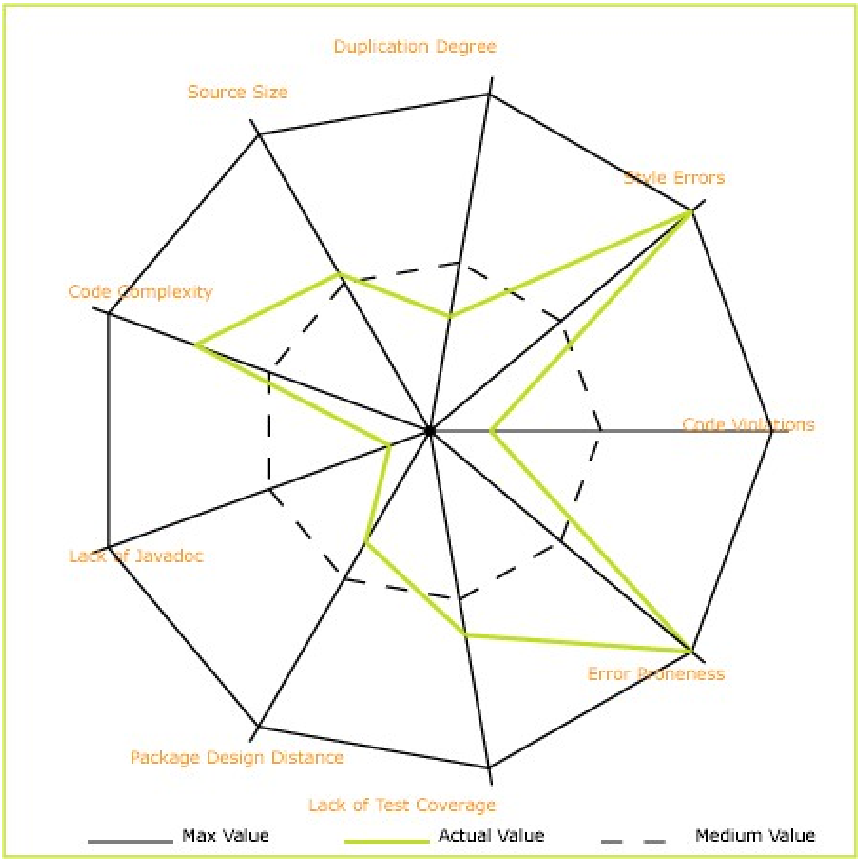


Figure 9.3: XRadar Spider Graph

XRadar has only a command line interface. It produces two types of HTML reports - one describing the "statics" of the system and one describing the "dynamics". XRadar Statics reports contain different views, such as a design, architecture, code and overview, among others. The views display different information, depending on the selected scope. One could drill-down in a javaDoc manner starting from the whole system, through its modules and packages to the separate classes.

Figure 9.3 illustrates a typical diagram of an XRadar Statics report, where a package is compared to all the other packages in the system on typical metrics in a spider graph. This view is available on all levels, except for the level of separate classes. The outermost line indicates the maximum values of the metrics. The dotted line indicates the average values of the metrics in the whole project and the green line shows the values for the selected package/module.

XRadar Dynamics reports display information about the development of the system through its releases. The same views are available as for XRadar Statics, but the information is displayed over time. The overview shows by means of graphs the values of the derived metrics "Total Quality", "Unit test suite",

”Architecture”, ”Design” and ”Code Quality” over time, i.e, for the different releases. Formulas for the calculation of those derived metrics are defined, as well as for the metrics they are based on. The ranges for the derived metrics are from 0 to 1. ”Design”, for example, is calculated as 0,2*(Number of Methods) +0,3*(Response for Class) +0,3*(Coupling Between Objects)+ 0,2*(Depth of Inheritance Tree), where Depth of Inheritance tree is the number of classes which have less than 5 superclasses, divided by the total number of classes.

For further details, please refer to the official documentation at [XRa09].

Metrics (Eclipse plug-in)

Metrics is a popular plug-in for Eclipse that calculates 23 of the most common source code metrics, such as McCabe’s cyclomatic complexity and Robert Martin’s architecture and structure metrics. The tool provides a visualization of the dependencies between packages. In figure 9.4 the package ”model” is selected. The packages, with which it is directly connected are displayed, as well as the direct dependencies among them. Note that the ”controller” package depends on 4 other (indicated by the number in the right corner) that are not displayed, since the focus is on the ”model”.

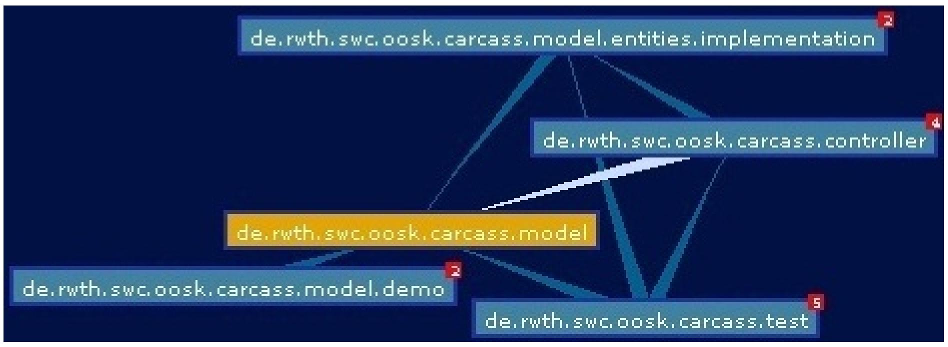

de.rwth.swc.oosk.carcass.model.entities.implementation
2
de.rwth.swc.oosk.carcass.controller
4
de.rwth.swc.oosk.carcass.model
de.rwth.swc.oosk.carcass.model.demo
2
de.rwth.swc.oosk.carcass.test
5

Figure 9.4: Metrics: Dependency Graph View [XRa09]

Other tools

SourceMonitor SourceMonitor is a lightweight tool, capable of analyzing C++, C, C#, VB.NET, Java, Delphi and Visual Basic code on a basic level. The supported metrics are very limited - only Lines of Code, Depth of Inheritance Tree, Cyclomatic Complexity and Number of Statements are measured, though the metrics are called differently. Lines of Code is calculated based on the actual source code. The results of the metrics can be visualized in spider charts (”Kiviat diagrams”). They can also be compared to the results of previous releases (”checkpoints”) in an additional graph.

JDepend [CC10] is a popular tool that calculates metrics, related to package dependencies. It is often used by other more advanced tools or in combination with other tools. JDepend does not collect source code complexity metrics. Afferent and Efferent Coupling are calculated only among packages, instead of among classes, as originally designed by Robert Martin. JDepend is often used along with **JavaNCSS**. JavaNCSS (stands for ”Non Commenting Source Statements”) [Lee09] measures standard metrics, including method cyclomatic complexity and javadoc length.

9.3 Discussion and Outlook

A wide variety of source code metrics tools exist on the market, both commercial and open source. They differ in the supported programming languages, the set of measured metrics, their calculation and interpretation. Beside the common metrics, the leading tools on the market offer advanced analysis features, based on complex derived metrics. Tools, such as SonarJ and XRadar among others, present a broader look into the analyzed system by means of advanced visualization and sorting functionalities.

The interpretation of the applied source code metrics by the different tools is not unambiguous and often not standardized. Discussions are going on whether the values of certain metrics are useful for making conclusions on the quality of the source code. A further problem is that some of the source code metrics do not scale, i.e., even if some ranges of values of such a metric have proven to be representative for smaller projects, they turn out to be wrong for larger-scale projects.

The large number of source code metrics tools available implies high interest in the field. As knowledge and experience aggregate with time, new dependencies are found that imply higher or lower code quality. Ranges for the values of metrics are specified and refined for different project sizes. Complex derived metrics are developed to enhance the insight into software systems and help pinpoint potential problems. Newer source code metrics tools offer advanced features, going far beyond mere counting.

9.4 Summary

The paper motivated the usage of source code metrics tools and attempted to demonstrate the reasons for their wide application in code refactoring, quality control and re-engineering. Some of the most common source code metrics were addressed, as well as their practical meaning for the disciplines mentioned above. The tools presented in this paper were divided into commercial and open source. The choice of the particular ones was made based on a search on the Internet, considering, among others, the customers of the tools, the tools' degree of acceptance in the industry (including user opinions), the range of functionality, the supported programming languages and usability of the tools. Mostly the features, which make a tool stand out, were presented.

Bibliography

[Ana09] CodeSWAT Analyst4j. Analyst4j overview. `http://www.codeswat.com/cswat/index.php?option=com_content&task=view&id=43&Itemid=63`, 2009.

[CC10] Inc. Clarkware Consulting. Jdepend. `http://clarkware.com/software/JDepend.html`, 1999-2010.

[Cor07] Microsoft Corporation. New for visual studio 2008 - code metrics. `http://blogs.msdn.com/fxcop/archive/2007/10/03/new-for-visual-studio-2008-code-metrics.aspx`, 2007.

[ea08] Olague et al. An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study. Journal of Software Maintenance and Evolution: Research and Practice 20 (3), 2008.

[Fle09] Andre Fleischer. Is quality measurable? `http://xradar.sourceforge.net/downloads/metrics-isQualityMesureable-de.pdf`, 2009.

[hel10] hello2morrow. Sonarj overview. `http://www.hello2morrow.com/products/sonarj`, 2010.

[iSA09] ARiSA Applied Research in System Analysis. Software quality metrics. `http://www.arisa.se/compendium/node88.html`, 2009.

[Lee09] Chr. Clemens Lee. Javancss - a source measurement suite for java. `http://www.kclee.de/clemens/java/javancss/`, 2009.

[LLC09] M Squared Technologies LLC. Rsm documentation center. `http://msquaredtechnologies.com/m2rsm/docs/index.htm`, 2009.

[S.A09] SMACCHIA.COM S.A.R.L. Ndepend. `http://www.ndepend.com/Default.aspx`, 2009.

[Sof09] Power Software. Krakatau metrics. `http://www.powersoftware.com/docs/justify/justify.pdf`, 2009.

[XRa09] XRadar. Xradar documentation. http://xradar.sourceforge.net/, 2009.

Chapter 10

Visualization of Hierarchical Data

Steffen Conrad

Contents

Abstract: Numerous methods are available for the visualization of hierarchical information. Treemaps are a graphical representation for trees of measurement data that can be used to display directory structures and organizational structures; hierarchical graphs are used to model the structure of software systems etc. The meaningful display of information gets problematic if the data structures get too large. Efficient and lightweight strategies are presented to overcome those problems, in particular adding shading to treemaps, and the use of an energy model for computation of meaningful layouts for hierarchical graphs which can be used for various analyses like questions about the structure of software systems, relationships between software entities, etc.

10.1 Introduction

A large quantity of the world's information is hierarchically structured – manuals, corporate structures, family trees, internet addressing, software system models, etc. Those hierarchical data structure can contain measurement data, for example size or weight attributes. It is very important to have useful techniques to visualize those measurement data structure, since humans have the ability to recognize the spatial configuration of elements in a picture and notice relationships between elements quickly. This visual ability allows people to grasp the content of a picture much faster then they can scan and understand text [Kam88]. Therefore a huge amount of visualization methods have been developed, this article will present two visualization techniques for trees and graphs.

Hierarchical structures are often represented by a graph, composed of nodes and links between these nodes. A widely used graphical representation for graphs and trees are different forms of node-link diagrams. In these diagrams elements of a structure are shown as nodes, relations between them are shown as links. Such graph and tree visualizations are very efficient and fast to understand. When those structures grow too large they can get confusing or do not offer any meaningful information anymore. It can also be difficult to find meaningful positions for each node and edge in a graph, so that the generated layout provides a useful view for a specific analysis. Big graphs also hardly fit on a screen or page while maintaining a significant size of nodes and links in the picture. To overcome these problems, this article will introduce two advanced visualization strategies, one being *cushion treemaps* that use shading to improve the display of complex tree structures, and an energy model for hierarchical graphs that allows the computation of meaningful graph layouts for different analysis purposes.

Section 10.2 will introduce cushion treemaps, a visualization for tree structures. The computation of meaningful layouts of hierarchical graphs will be discussed in section 10.3. A short discussion about the presented topics and their use in applications will be presented in sections 10.4 and 10.5.

10.2 Cushion Treemaps

A wide known traditional graphic representation for trees is displaying the root node at the top and its child nodes below connected by lines [Knu97]. A different visualization concept named *tree-maps* has been developed by Ben Shneiderman at the University of Maryland [Shn92], [JS91]. He was looking for a better representation of the storage usage of a file system, trying to identify big files and directories that could be deleted to free up space on his hard drive. Since most file systems are organized as a tree, he tried to find a way to visualize the sizes of each directory in that tree structure. The approach of displaying the tree in its traditional graphical form with its sizes as an attribute at the nodes did not work very well. Even for small file systems the screen space was too limited. This made it very hard for him to grasp the entire picture.

10.2.1 Treemaps

Shneiderman solved his problem with a 2-d space-filling approach, recursively subdividing a rectangular display space according to the size of subtrees for a given tree node. Starting with the root node of the tree for the full rectangular display space, it gets vertically split into pieces according to the number of child nodes. The size of each pieces gets chosen proportionally to the size attribute of the according child node (i.e. the summarized sizes of its child nodes). For each of those child nodes the partitioning of the according rectangle is repeated for its own child nodes. Hereby the algorithm alternates between vertical and horizontal partitioning for each level of the tree.

Figure 10.1 shows the resulting treemap representation of a tree. The size attribute of each tree node is contained in the node name, e.g. the root node A16 is of size 16. It is represented by the outermost rectangle in the treemap. This rectangle contains all other rectangles, since all other tree nodes are contained in the subtrees of the root node.

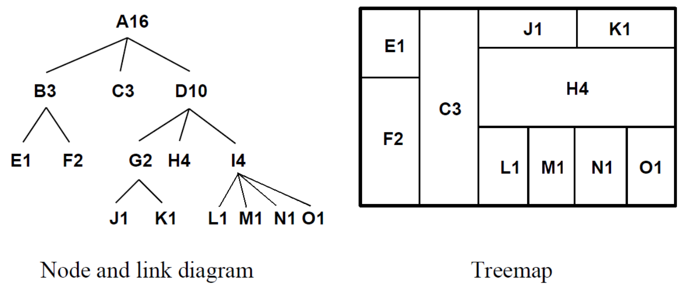


Figure 10.1: Treemap of a small tree structure

Tree-Map Algorithm

The layout algorithm assumes a tree structure and a rectangular area defined by uper left and lower right coordinates *P1(x1,y1), Q1(x2,y2)*. The number of childs of the current tree node determines the number of partitions this rectangular area needs to be split into. Since the size of a child node is a fraction of the size of its parent node, the partitions of the child nodes *i* are split proportionally to their size in relation to the size of the parent. The algorithm is then recursively called for each child node and its according rectangular area, alternating between horizontal and vertical splitting. By setting the *axis* argument to zero, the initial splitting will be done vertically.

```
Treemap(tree_node, P[0..1], Q[0..1], axis)
  Paint_Rectangle(P, Q);       -- paint full area
  width := Q[axis] - P[axis];  -- compute location of next slice
  for i := 1 to tree_node.num_children do
    -- compute size of slice
    Q[axis] := P[axis] + (Size(child[i])/Size(tree_node))*width;
    -- paint the child nodes recursively,
    Treemap(child[i], P, Q, 1 - axis);
    -- skip position to the next slice
    P[axis] := Q[axis];
  endfor
```

The described algorithm runs linearly with the number of nodes in the tree. The size of each node therefore can be computed beforehand in linear time and each run of *Treemap()* is done once for each tree node. Figure 10.2 shows a colored treemap of a file system containing 1000 files. The large files can easily be identified.

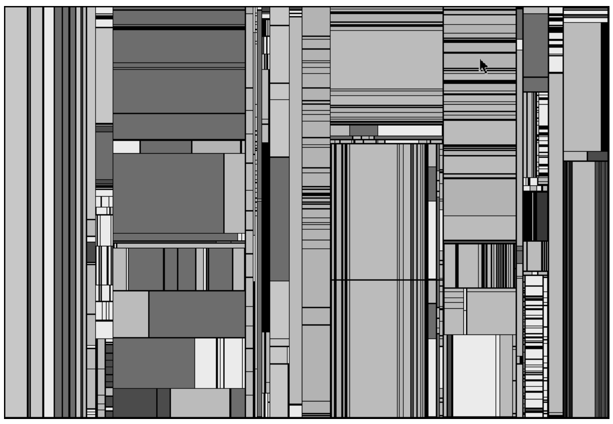


Figure 10.2: Treemap of a file system with 1000 files [JS91]

10.2.2 Cushion Treemaps

Treemaps also have limitations, such as not being able to provide a meaningful visualization of the structure of a tree under certain conditions. In its worst case, a balanced tree where every parent node has the same number of children, the treemap degenerates into a regular grid. Another problem is related to the boundaries between subtrees. While leaves of a subtree are still displayed close to each other, rectangles displayed close to each other in the treemap can be assigned to very different subtrees in the tree structure.

As an example, the top left treemap in figure 10.4 on page 156 shows an organization chart, modeled after the employment structure of an university. Important questions about the size of a section or if the division into units is balanced or not are hard to answer from the treemap. While it is possible to use different colors, or different line sizes for each partition of the treemap, the required information would still be hard to track for a user. An alternative is to use shading to visualize the structure of the tree. The concept of *cushion treemaps* will therefore be introduced.

A simple Shading Model

The structure of the tree displayed in the tree map can be emphasized by adding shapes to the treemap. The human visual system is trained to interpret variations in a shade as illuminated surfaces, so the aim is to construct a surface which shape is sufficient to encode the tree structure. The solution is introduced by a one-dimensional example, the binary subdivision of an interval. This is then generalized to a two-dimensional method afterwards.

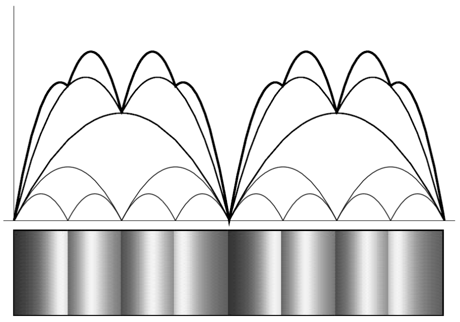


Figure 10.3: Binary subdivision of interval [WW99]

An interval gets subdivided into two halves, adding a bump to each subinterval. Done recursively for three levels, the division results in eight segments and the top-most curve as seen in figure 10.3. By interpreting this curve as the side view of a bent strip and render it as viewed from above it forms a sequence of ridges. Each of the segments is clearly visible, bounded by the sharp continuities in the shading. Furthermore the binary tree structure is also emphasized, as the depth of the valleys between the segments is proportional to the distance between segments in the tree [WW99, p.3].

The idea of the bent strip can be generalized to a two-dimensional case, resulting in a ridge formed like a parabola. These ridges are then applied to each rectangle of the treemap. The visualization can be controlled by two parameters, the height h and a scale factor f [WW99, p.2-3]

After applying the ridges to the treemap, each ridge gets shaded by a simple model like diffuse reflection [JDFH96]. Results of this method are shown in figure 10.4. Here the tree structure can be identified without effort. The high scale factor in the top right treemap emphasizes the details of the tree, the lower scales in the bottom left and right treemaps show the global structure of the tree.

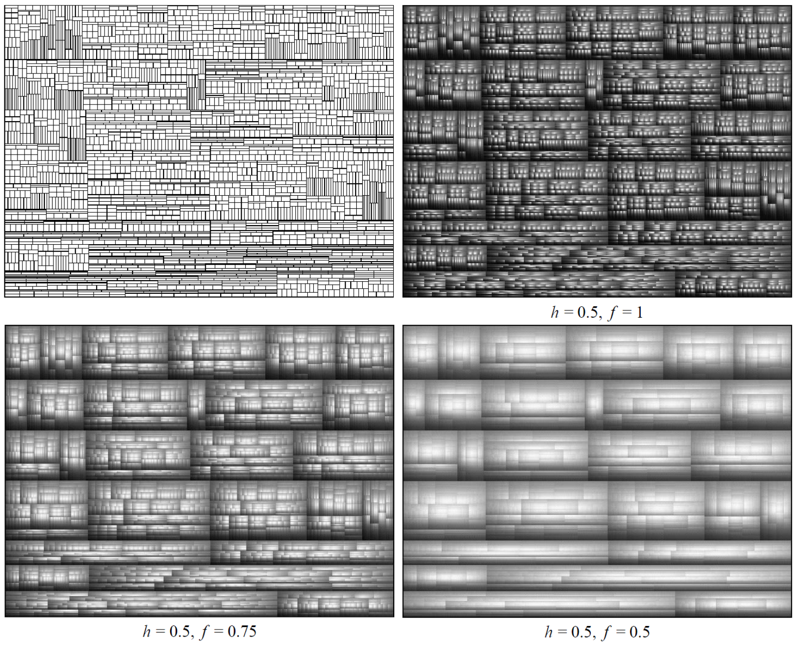


Figure 10.4: Cushion treemaps of organization [WW99]

Adding cushions to the Tree-Map Algorithm

The above method can easily be added to the presented treemap algorithm. Instead of only painting the border of the rectangle for each segment of the tree, a cushion needs to be added to each rectangle, which then gets rendered by the diffuse reflection shading model. The source code of the resulting algorithm is very compact and still scales linearly with the size of the tree and the size of the display space [WW99, p. 4-5].

10.3 Layout Styles for Hierarchical Graphs

Hierarchical graphs are widely used when modeling the structure of software systems – entities like packages, classes, methods or attributes get represented by graph nodes; relations between entities like inheritance, method calls or attribute access get represented by graph edges; containment hierarchy of the entities can be modeled by a hierarchy tree over the graph nodes.

To support comprehension, evaluation and improvement of the structure of large software systems, views on different levels of abstraction are required; like visualizing relationships between packages on a global level, displaying the interaction of methods and attributes of a few classes in detail, or views including different levels of detail to show methods and attributes of a class in their global context. The visualization of such views requires the computation of layouts, in example the positions and size of graph nodes in a two- or three-dimensional space. Therefore Andreas Noack and Claus Lewerentz at the Brandenburg University of Technology in Cottbus developed requirements for meaningful layouts that support these analyses, and formalized these requirements to automate the computation of the corresponding layouts [NL05].

After a brief introduction to hierarchical graphs, a space of layout styles will be introduced which organizes layouts along three dimensions – the degree of clustering, hierarchicalness and distortion. These requirements get combined into an energy model that allows the automatic computation of graph layouts that highlight specific characteristics of a graph of a software system.

10.3.1 Hierarchical Graphs

A *hierarchical graph H* consists of a directed graph *G* and a rooted tree *T*, with the leaves of *T* exactly being the nodes of *G*. The tree *T* is called the *hierarchy tree* of *H*, *G* is called the *underlying graph* of *H*. The left side of figure 10.5 shows a hierarchical graph and its corresponding hierarchy tree. The nodes of the hierarchical graph are represented as boxes, the edges of the underlying graph are displayed as lines with arrows, and the edges of its hierarchy tree are represented as nesting boxes.

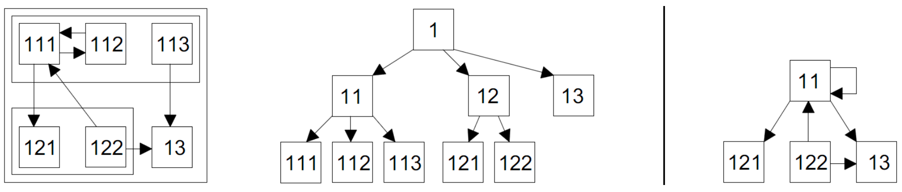


Figure 10.5: a hierarchical graph, its hierarchy tree, and a view [NL05]

The advantage of hierarchical graphs in comparision to normal graphs are that *views* can be defined on those graphs. A view utilizes a collection of nodes of the hierarchy tree and uses them to display the information on these nodes as they would represent all of their child nodes of the underlying graph, including their links. This allows to display a summary or abstraction of the underlying graph. The right part of figure 10.5 shows a view on the graph from figure 10.5. Here the node 11 is used to represent the nodes 111, 112, and 113; all edges adjacent to these nodes are displayed adjacent to node 11 instead.

Such views are applicable for a wide amount of visualizations, for example focusing on a single package in a dependency graph to display its relations to other packages by simply showing the package itself in full detail and displaying each related package as a single node only. It would be easy to spot which class of the package is interacting with the outside system etc.

10.3.2 An Energy Model for Graph Layouts

To compute layouts for the hierarchical graphs that serve specific analysis purposes, requirements for these layouts must be defined. To determine if a layout meets the analysis requirements, an energy model is used to formalize the quality. It assigns an energy to each layout, specifying the quality of node positions and sizes according to an analysis requirement. *Better layouts* lead to smaller energies, so the finding of good layouts for a specific analysis purpose results in finding a layout with minimum energy.

Perfect energy minimization algorithms that can be used to compute those layouts by finding a global minimum are known, but are not useable in a practical way because of their complexity. Instead there are several proven heuristics that reliably and efficiently find layouts with low energy. For the presented examples, an algorithm that was introduced for the simulation of astrophysical systems has been used [BH86]. It has already been applied on the computation of graph layouts by Quigley and Eades in 2001 [QE01].

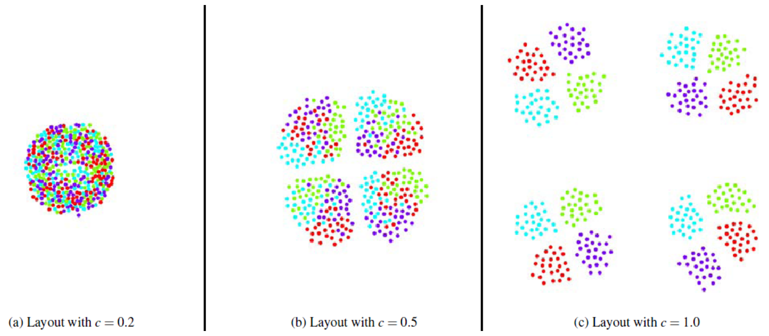
The following sections will introduce the analysis requirements and iteratively formalize them into an energy model. They have been presented by Noack and Leverentz in *A space of layout styles for hierarchical graph models of software systems* [NL05]. It has a strong focus on using the presented strategies to find suitable layouts for different software system analyses.

Degree of Clustering

The first requirement for graph layouts presented is the *degree of clustering*. It models a tradeoff of layouts with locally interpretable distances versus layouts with globally interpretable distances. This allows analysis questions about single entities and relationships on the one hand, and questions about the global structure of a software system on the other hand.

The clustering is dependent on edges between the nodes of the underlying graph. Connected nodes attract each other in a clustered layout, non-connected nodes repel each other. The more of a group of nodes are connected to each other, the more likely the are belonging to the same cluster in a resulting graph layout. While a small degree of clustering allows local analyses on the neighborhood of software entities (clusters do not get grouped together so a decent level of detail remains), a high degree of clustering focuses on the global structure (placing strongly connected groups of nodes together, thus highlighting connections between several clusters).

The influence of the clustering parameter c is best shown by a small pseudo-random graph that is built out of 4 clusters with 4 subclusters. While each node of a subcluster is connected to every other node of the subcluster, only some nodes of different subclusters of a cluster are connected. Even less nodes of different clusters are connected. Different layouts of this clustered graph are shown in figure 10.6 – while the left layout does not focus on clusters, the middle and right graphs highlight the different clusters of the graph.



(a) Layout with $c = 0.2$ (b) Layout with $c = 0.5$ (c) Layout with $c = 1.0$

Figure 10.6: random graph with clusters (edges hidden) [NL05]

Non-clustering analyses are composed of questions like which other methods are calling a specific method – if this method signature would change, all invoking methods would have to adapt those changes. In a more high-level manner, questions about packages could be answered. An architecture could require that a certain package *p1* may only be used by packages *p2* and *p3*, but *p1* not using the packages *p2* and *p3*. Looking for all packages that use the package *p1* could easily answer if this rule is violated or not.

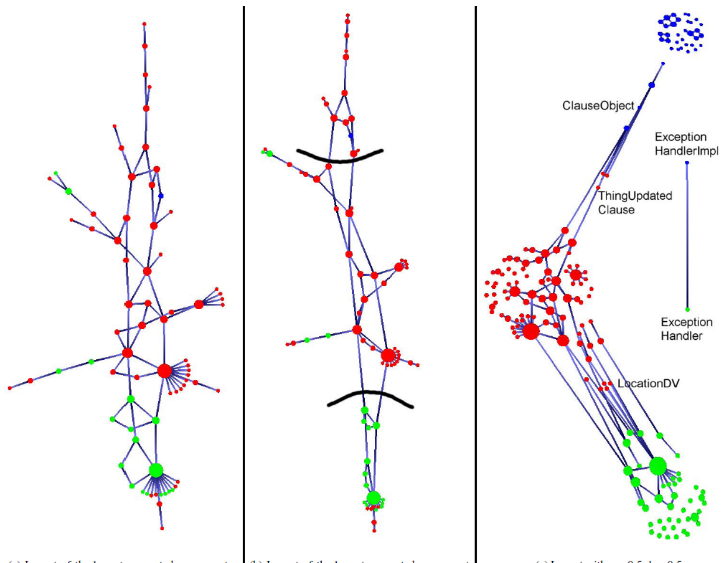
Analyses related to a high level of clustering are decomposition questions, focusing on breaking a system down into groups of entities that are strongly related to each other, while only loosely related to entities in other groups. This is also called software clustering, which tries to find interfaces for these groups of entities to the rest of the system.

Degree of Hierarchicalness

The *degree of hierarchicalness* adds the containment hierarchy of software systems to the layout space. By not focusing on the hierarchy, the layout reflects the layout of the underlying graph, placing adjacent nodes closely. The opposite, focusing on the hierarchy, the layout reflects the hierarchy tree, placing nodes with common parents, grandparents etc. close to each other. When focusing on the containment hierarchy, various questions about package and class hierarchies can be answered.

The clustering and hierarchy layouts get combined, making those layouts useful for comprehension, evaluation and improvement of software architectures. Therefore the energy model of section 10.3.2 gets altered, adding a second dimension to the layout styles, called the degree of hierarchicalness h. It reflects the gravitation between nodes that are related in the hierarchy tree [NL05, p.159].

The combination of clustering and hierarchy layouts can be demonstrated on the JWAM package, an open source framework for the construction of distributed software systems [JWA04]. Figure 10.7 shows layouts with different clustering levels on the left and middle, adding focus on hierarchy to the layout on the right, now also grouping nodes in respect to their containment hierarchy. The layout shows the three different subpackages of *jwam* – *jwam.handling* (red), *jwam.technology* (blue), and *jwam.lang* (green). The only relation between the top and bottom package is the inheritance of *ExceptionHandler* and *ExceptionHandlerImpl*. The dependency between the packages could easily be resolved by moving them to the same package.



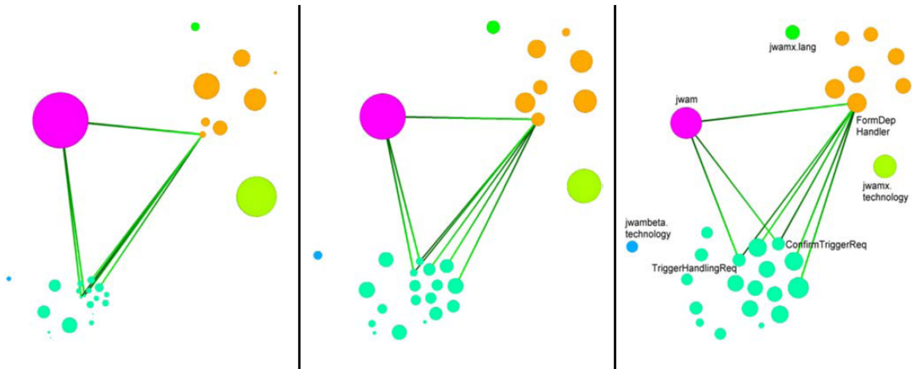
(a) Layout of the largest connected component with $c = 0.2$ (b) Layout of the largest connected component with $c = 0.5$ (c) Layout with $c = 0.5$, $h = 0.5$

Figure 10.7: relationships between classes in the *jwam* package [NL05]

Degree of Distortion

Layouts reflecting clustering and hierarchy rely on weights assigned to their edges which reflect the number of edges of the underlying graph they represent. While this is no problem for most of the analysis tasks, in some layouts every single relationship between entities needs to be emphasized in the same way. This could be required when a specific class or attribute needs to be shown in its relation to the rest of the system. It may be a small part of the system, but a very important part of the analysis task. So while analysing local details in their global context, *distortion* can be used to magnify details that otherwise would be hidden.

The level of distortion d therefore builds the final dimension of the space of layout styles. It gets integrated into the already existing layout energy model, influencing the weightings for nodes and edges [NL05, p.161]. While a low level distortion means that the weightings used in the energy model remain unchanged, every edge has the same weight on a high-distortion level.



(a) $d = 0.0, c = 0.5, h = 0.9$ (b) $d = 0.5, c = 0.5, h = 0.9$ (c) $d = 1.0, c = 0.5, h = 0.9$

Figure 10.8: detail views in *jwam* with different levels of distortion [NL05]

The *jwam* example in figure 10.8 shows a mutual dependency, namely method calls from *jwambeta* to *jwamx* and from *jwamx* to *jwambeta*. This generallly complicates changes, understanding and reuse, also the package names indicate that this violates the intended architecture. The three layouts visualize this problem on class level with varying clarity, showing that the dependency corresponds to calls from *FormDepHandler* to *TriggerHandling* and *ConfirmTriggerReq*. The views in those layouts itself contain different levels, for example only showing a single node for the *jwam* package, and only showing some *classes* or the other packages to only focus on the involved entities. A higher distortion in the right layouts provides a much better vision on details to evaluate according changes to remove the mutual dependency.

10.4 Tools

SequoiaView – disk browsing with cushion treemaps

Van Wijk and van de Wetering not only presented the cushion treemap concept, they also demonstrated its capabilities in a tool called *SequoiaViews* [Seq02]. It parses a given directory, calculating file and directory sizes, and then renders a cushion treemap out of it. This way, the visualization of the directory structure can be navigated and analysed.

The implementation showed that there are still layout problems when generating treemaps on directories with a lot of small files – the corresponding display degenerates into groups of adjoining lines that do not provide any useful information. Van Wijk and van de Wetering solved this by enhancing the layout algorithm to *squarified cushion treemaps*, using squares to layout the files on each level. While this strategy adds some complexity to the layout algorithm, the results are much better. The screenshot in figure 10.9 shows this layout of C:\Windows on Windows XP.

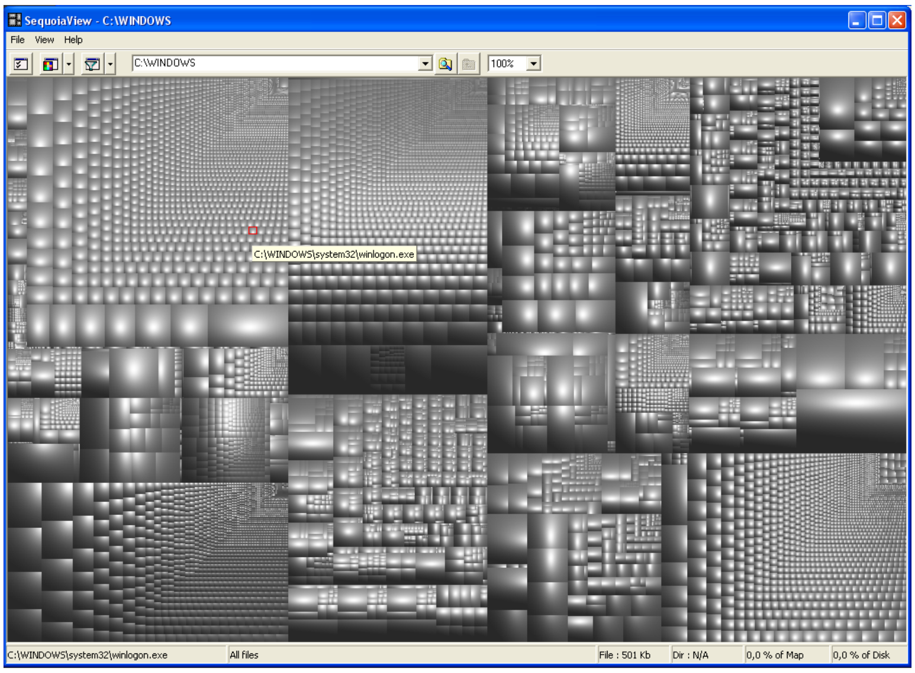


Figure 10.9: SequoiaView displaying C:\Windows

Software landscapes – a graph visualization method

The research team working on the energy model based graph layouts have explored many more graph visualization techniques in several projects, for example *CrocoCosmos* [Cro03], a tool for analysis and visualization of structural and metrics information of large object oriented programs. Another project worth mentioning is a collaboration with a team of the University of Konstanz about the use of *software landscapes* to visualize large software systems. It combines the presented layout styles with a different graph visualization technique, using a landscape metaphor to provide intuitive navigation and comprehension [BNDL04]. As an example a layout method called *Hierarchical Net* is shown in figure 10.10. It displays a part of the JWAM package [JWA04].

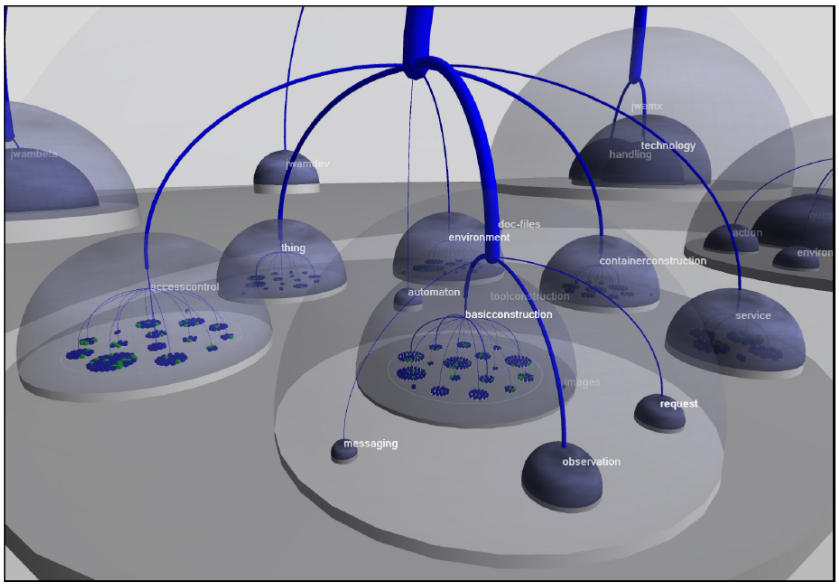


Figure 10.10: hierarchical net view of the JWAM package [BD04]

10.5 Discussion

Treemaps and their variations can support various software analyses, especially when displaying measurement data. They are restricted to tree structures, which limits their usefulness when working with complex data structures. *Cushioned treemaps* and *squarified cushion treemaps* solve the problem of displaying complex tree structures of measurement data.

The energy model for graph layouts provides styles for views of hierarchical graphs that can be organized in a space with three dimensions. By altering the parameters and defining an appropriate view on the underlying graph of a software system, a user can easily create graph layouts to aid in several analysis questions. Hereby the parameters do not need experimental tuning, each parameter value has a clear interpretation. The well-defined correspondence between properties of the layouts and properties of the graph allows to draw valid inferences about a software system directly from its corresponding graph layout. This characteristic has largely been missed in previous energy models.

Treemaps and the energy model based graph layouts can also be combined, for example displaying measurement data of packages of a software system in treemaps, while displaying relations between different packages in a graph based layout.

The two presented methods help to enhance existing measurement data visualization concepts to improve their clarity and useability. While cushion treemaps try to solve the problem of visualizing complex measurement data hierachies, the presented energy model of the second section addresses the support of various software system analysis tasks with appropriately computed graph layouts. Both concepts have the potential to improve software system analyses, they offer intuitively useable software metrics visualization techniques.

Bibliography

[BD04] M. Balzer and O. Deussen. Hierarchy based 3d visualization of large software structures. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 598–599. IEEE Computer Society, 2004.

[BH86] J. Barnes and P. Hut. A hierarchical o(n log n) force calculation algorithm. *Nature 324*, pages 446–449, 1986.

[BNDL04] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz. Software landscapes: Visualizing the structure of large software systems. In *Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 261–266. Eurographics Association, 2004.

[Cro03] CrocoCosmos – visualization of object-oriented programs, 2003. http://www-sst.informatik.tu-cottbus.de/CrocoCosmos/index.html.

[JDFH96] Foley J.D., A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics – Principles and practice, 2nd edition*. Addison Wesley, 1996.

[JS91] Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 284–291. IEEE Computer Society Press, 1991.

[JWA04] JWAM, a framework for the construction of distributed systems, 2004. http://sourceforge.net/projects/jwamtoolconstr/.

[Kam88] T. Kamada. *On visualization of abstract objects and relations*. PhD thesis, Dept. of Information Science, Univ. of Tokyo,, 1988.

[Knu97] Donald E. Knuth. *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997.

[NL05] Andreas Noack and Claus Lewerentz. A space of layout styles for hierarchical graph models of software systems. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 155–164. ACM, 2005.

[QE01] Aaron Quigley and Peter Eades. Fade: Graph drawing, clustering, and visual abstraction. In *GD '00: Proceedings of the 8th International Symposium on Graph Drawing*, pages 197–210. Springer-Verlag, 2001.

[Seq02] SequoiaView – a disk browsing tool utilizing cushion treemaps, 2002. http://www.win.tue.nl/sequoiaview/.

[Shn92] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

[WW99] J.J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*, pages 73–78. IEEE Computer Society, 1999.