TOWARDS ANCHORING SOFTWARE MEASURES ON ELEMENTS OF THE PROCESS MODEL

Bernhard Daubner and Bernhard Westfechtel
Bayreuth University, Applied Computer Science I
95440 Bayreuth, Germany

Andreas Henrich
Bamberg University, Media Informatics
96045 Bamberg, Germany

Keywords: Automatic software measurement, anchoring software measures, process models, measurement tool, maven.

Abstract: It is widely accepted that software measurement should be automated by proper tool support whenever possible and reasonable. While many tools exist that support automated measurement, most of them lack the possibility to reuse defined metrics and to conduct the measurement in a standardized way. This article presents an approach to anchor software measures on elements of the process model. This makes it possible to define the relevant software measures independently of a concrete project. At project runtime the work breakdown structure is used to establish a link between the measurement anchor points within the process model and the project entities that actually have to be measured.
Utilizing the project management tool *Maven*, a framework has been developed that allows to automate the measurement process.

1 INTRODUCTION

Effective management of software development projects requires permanent assessment of both the actual projects and the underlying development processes. Managers need to control projects quantitatively in order to maximize estimation accuracy based on historical project data, to minimize risks and time-to-market, and to reliably reproduce the related processes (Auer et al., 2003). As the continuous collection and analysis of measurement data is crucial in tracking and managing software development processes efficiently, the measurement process must be automated by proper tool support whenever it is possible and reasonable.

Today many tools are available to compute certain measures about software artifacts like source code, to store measurement data in a database and to create the corresponding reports. Also general frameworks and guidelines have been proposed to integrate software measurement into the software engineering environment (Basili and Rombach, 1988), (Kempkens et al., 2000), (Münch and Heidrich, 2004). Finally there exist measurement approaches like *GQM* (Basili et al., 1994) and *PSM* (McGarry et al., 2002) that support the project manager to define what measures to collect.

Within the above mentioned measurement frameworks it is often stressed that models and metrics defined for the whole organization should be reused to make measurement results comparable and to build up an experience base that can be used to assess future projects. In order to assure comparability of measurement results, measurement must be integrated into the software development process for a new project before project execution starts (Kempkens et al., 2000).

Concerning these last two aspects we think that there exists no proper tool support that provides both automated software measurement and reuse of already defined metrics. Moreover, it should be possible to implement the measurement program on top of an existing software development process. Hence, we suggest to anchor the software measures on elements of the applied process model. Thus the software measures that have to be collected can be determined *ex ante* and independently of a concrete project. This allows to define in advance, which software measures have to be computed on every software development project in order to make these projects comparable.

We will show within the next section that this approach is fairly new since the other published approaches either lack tool support for automated mea-

Daubner B., Westfechtel B. and Henrich A. (2006).
TOWARDS ANCHORING SOFTWARE MEASURES ON ELEMENTS OF THE PROCESS MODEL.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 232-237
DOI: 10.5220/0001318502320237

surement or are based on a special process model that is needed in order to provide automatic measurement.

2 RELATED WORK

The need for software measurement has been widely accepted and there exist many commercial software measurement tools. In (Auer et al., 2003) a survey on up-to-date measurement tools has been conducted. Only two out of five products were able to collect measurement data automatically. These tools (*MetricCenter*[1] and *ProjectConsole*[2]) mainly focus on software measures concerning the whole project (number of requirements, number of defects, etc.). These kinds of tools are certainly useful in order to provide a management view on ongoing projects and are also described in terms of *project dashboards* (Selby, 2005). But we think that they hardly provide support for reuse of the defined metrics to enable more fine granular comparisons between projects. This aspect is addressed by the *Software Project Control Centers* decribed in (Münch and Heidrich, 2004) including a reference model of concepts and definitions around SPCCs. A tool support for the proposed architecture however is not yet available.

In (Lott, 1996) a framework is presented to integrate measurement and process models in a way that supports automation of “measurement-based feedback”. Automated support for measurement-based feedback means that software developers and maintainers are provided with on-line and detailed information about their work. Lott has realized his approach by implementing a dedicated process-centered software engineering environment. Hereby the system informs the users about the tasks that they are expected to perform and collects data that are associated with the realization of these tasks. However most of the data is not measured automatically but must be provided by the user by means of a form-based interactive tool.

Also the *APEL (Abstract Process Engine Language)* (Dami et al., 1998) has been developed as process modeling and controlling framework with integrated measurement. It allows the user to link the measurement model with the process model in order to control which measures should be collected at what stages in the process. Since APEL acts as a workflow engine the automatic support it provides only works if the processes are exactly executed in the way they are specified.

In (Kempkens et al., 2000) a framework for integrated tool support within measurement programs is presented, which gives guidelines for setting up measurement tool support for software development processes. Hereby the framework allows companies to use their existing tools and processes. Within the presented case studies however much of the data collection was done manually and no evidence has been given for the demanded ”reuse of models and metrics”.

Finally, a contrary approach has been presented by Johnson within his conference paper ”You can’t even ask them to push a button: [...]” (Johnson, 2001), where he introduced the approach of his tool *Hackystat*. The Hackystat software uses sensors to gather the activities of each user within the software development environment. Thus it analyses and logs what kind of work (programming, testing, modeling, etc.) the user is just performing. Hereby the system acts unobtrusively and the user is in no way interrupted within his work. It is however not easy to interpret the vast amount of data collected by Hackstat in a meaningful way.

3 ANCHORING SOFTWARE MEASURES

3.1 Software Measures in the Context of Process Models

Our aim is to describe a standardized way to integrate software measurement into the software development process in order to make software development projects comparable with respect to the applied software measures and the way the software measures are collected.

The crucial point is that especially those software measures that are relevant to the project at management level are fixed independently of the concrete project. Software measures that generally have to be collected are for instance determined at the introduction of a company wide measurement campaign (McGarry et al., 2002), are derived from tactical or strategical company goals with frameworks like the *Goal Question Metric Paradigm* (Basili et al., 1994) or are determined by a process evaluation methodology like the *Capability Maturity Model Integration (CMMI)* (Chrissis et al., 2003).

This insight leads directly to our approach to anchor the software measures on elements of the process model, because the process model is the basis that all the projects we want to measure will build on.

The process model structures the software development process into units typically called phases, disciplines or activities. Here we can distinguish between fine granual process models like the *V-Modell-*

[1] http://www.distributive.com

[2] http://www.ibm.com/developerworks/rational/products

XT[3], which serves as the standard process model in Germany for managing government IT development projects, and more "pragmatic" process models like the *Rational Unified Process* (Kruchten, 2003) that contain guidelines about the workflows to operate on. One interesting question in this regard is how certain work efforts are distributed over the activities. For example one would expect that most of the programming effort is spent on activities belonging to the implementation phase. But is is not unlikely that programming efforts also are carried out within the scope of analysis related activities if requirements have to be clarified by means of prototypes. It might even be possible, if the development is accomplished in an agile way, that most of the programming effort is accumulated during analysis activities.

In order to monitor this we suggest to anchor the appropriate size measures on the relevant elements of the process model. Thus, we get software measures like

- LOC within the scope of programming related activities and
- LOC within the scope of analysis related activities

These measures use elements of the process model as anchor points and not concrete artifacts.

3.2 The Work Breakdown Structure

This yields to the demand to identify the artifacts that have been created or modified within the scope of a certain activity of the process model in order to measure their sizes respectively. For this purpose we utilize the work breakdown structure of the project as link between the process model which the project is based on and the above mentioned artifacts.

The work breakdown structure (WBS) (Project Management Institute, 2001) represents the structure of a project and contains the essential relationships between the elements of a project. The project is structured in a hierarchical manner into subtasks and work packages. This leads to a tree structure as shown in Figure 1, which contains all project activities that have to be performed within the individual development phases.

Identifying Artifacts Utilizing Wbs Codes

Since the WBS is based upon the activities of the process model, we can assume that for each activity of the process model a corresponding work package exists within the work breakdown structure. Thus we can reduce the above demand to identify the artifacts associated with a certain activity to the determination of the artifacts that have been created within a certain work package. This however is possible by means of the identifying key (*WBS Code*) that usually each element of the work breakdown structure is associated with (Project Management Institute, 2001). Using for instance a *decade code scheme* (Figure 1) the number of digits unequal to 0 within the WBS code determines the position of the WBS element within the hierarchical work breakdown structure.

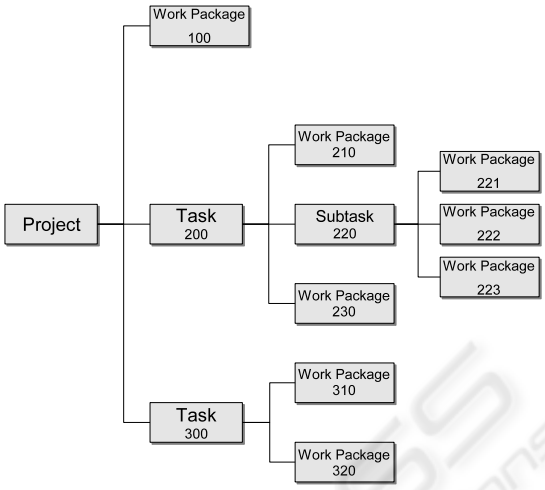

Figure 1: Work Breakdown Structure with decade code scheme.

The WBS codes not only identify the work packages within the work breakdown structure. They also can be used to label the artifacts that have been created or modified within a particular work package. For that purpose it is necessary to maintain all artifacts under version control. Whenever a *commit* statement for a modified version of an artifact is executed within the software configuration management (SCM) system, the WBS code of this work package has to be stored together with the change-log information.

Thus by means of the WBS code within the change-log information of the SCM system the artifacts and also the changes of the artifacts are associated with the corresponding activities of the process model. And software measures that have been anchored on elements of the process model can be applied respectively.

Cross-Project Software Measurement

Since companies usually perform several similar software development projects, in the course of the time a standard work breakdown structure is established based on the company-specific process model. The standard WBS serves as basis for the concrete work breakdown structures used in the individual projects (Futrell et al., 2002). For the latter this standard work breakdown structure has to be extended or shortened respectively.

[3] http://www.v-modell-xt.de

The common elements of the WBS however keep hold of the same WBS code. Utilizing this standard work breakdown structure makes the application of software measures anchored on the process model also in a cross-project manner possible. Those software measures that are associated with the company-specific process model can be collected by means of the cross-project uniform WBS codes of the corresponding subtasks and work packages.

3.3 Agile Software Development

In Section 3.1 we have explained our approach considering well defined process models with a clear and elaborated structure. This immediately leads to the question whether this approach also is applicable on less elaborated or less process based development methodologies. Particularly within agile development methodologies it is not very common to establish a work breakdown structure (Beck, 2004), (Cockburn, 2001) that our approach needs to identify the artifacts.

The apparently obvious idea to collect software measures within the scope of the individual iterations of the agile development process does not lead to any benefit in order to compare several development projects or to assess the progress of the ongoing project.

But also software projects using an agile development style can be structured into at least two dimensions. First of all there are several kinds of activities to distinguish that are performed during a development project. Ambler for example mentions in the style of the RUP the "agile disciplines" *Modeling*, *Implementation*, *Test*, *Deployment*, *Configuration Management*, *Project Management* and *Environment* (Ambler, 2002). In addition to that the software units that are created can be categorized according to their function. Here one can think of *application logic*, *presentation*, *data management*, *transaction management*, *logging* etc. . This means that the activities of an agile software project can be mapped into a matrix with the axes *discipline* and *function*. Therefore also the artifacts produced within an agile project can be tagged within the SCM system with 2-tupels that inform about the discipline they have been created in and the function they have.

Thus, regardless of the existence of a work breakdown structure, the artifacts under version control and the time records within the time recording system can be tagged with codes that identify the implemented functionality and the discipline within that scope the effort has been performed. These codes provide a "generic project plan" and we can use the elements of this "generic WBS" to anchor our software measures on. In Section 4.3 we will demonstrate this agile application of our measurement approach on a small development project.

3.4 Preferred Usage of Our Approach

We think that this measurement approach is preferably suitable for measures that are relevant for project controlling. Here often the measures *size*, *time*, *effort*, *defects* and *productivity* are mentioned (Russac, 2002), (Putnam and Myers, 2003). For these measures our approach supports automatic measurement and reuse.

The only thing our approach needs is that all projects are based upon a consistent skeletal structure. This can be a standard WBS for process-based development projects or a discipline-function-matrix for agile projects. The skeletal structure provides the numbering scheme in order to tag the produced artifacts within the SCM system or administrative information like time records within the time recording system.

4 INTEGRATING AND AUTOMATING SOFTWARE MEASUREMENT

4.1 The Project Management Tool Maven

In order to continuously measure the software development process and to get the required feedback on process improvement attempts the collection of the software measures has to be automated and must be integrated into the development process. Implementing our approach we have restricted ourselves to Java projects. This allows us to use the software *Maven* from the *Apache Project*[4]. Maven is a tool that supports the management and the development process of Java projects. On the one hand it supports the developer at the so called *build process*, i.e. Maven compiles the source code and considers thereby dependencies like additional libraries that have to be made available first. With this respect Maven can be regarded as an alternative to *Ant*[5].

In addition to that it provides means to hold important information about project members, project ressources (e.g. documents, licences) and about the software configuration and bug tracking systems that are used within the project. Using plugins, the functionality of Maven can be extended anytime. The plugins thereby can access the project information and thus for instance access the software configuration

[4] http://maven.apache.org

[5] http://ant.apache.org

system in a generic manner without the need to know the exact type of the SCM software.

4.2 Measurement Transmitters for Software Measures

At this point we join and use Maven plugins in order to implement what we call *measurement transmitters*. In climate research measurement transmitters are used for instance to measure rainfall. A software measurement transmitter however is a Maven plugin that is capable of computing a certain software measure. For example a `JavaLOC` measurement transmitter is able to determine the *Lines of Code* of Java program files. And as a rainfall measurement transmitter can be set up at different locations the `JavaLOC` measurement transmitter can likewise be configured in that way with the Maven XML configuration file that only such code lines are counted that have been created within the scope of a certain element of the process model.

We do this by anchoring the measurement transmitter on the corresponding work package of the WBS using the XML configuration file of the plugin.

Querying the software configuration management system, which also is referenced within the Maven configuration, and utilizing the change-log information, it is possible to identify such source code files (including the revision numbers) that have been created or modified within the scope of the mentioned work packages. Now the affected versions of these source files successively have to be checked out in order to measure their changes in size in LOC.

Measurement transmitters also can be combined. For example a `JavaLOC` measurement transmitter can be combined with an `Effort` measurement transmitter which leads to a simple `Productivity` measurement transmitter. First the `JavaLOC` measurement transmitter has to determine the code size. Then the `Effort` measurement transmitter determines the expenditure of time by querying the time recording database. The results refer to the same work packages. Thus, the productivity during the treatment of these work packages can be computed.

4.3 The Maven Measurement Framework

We call our implementation of this measurement approach on the top of Maven the *Maven Measurement Framework (MMF)*. It is a framework because it consists of a collection of API functions that are invoked by Maven plugins which thereby implement the measurement transmitters. The API provides the application logic like analysing the change-logs of the SCM system, querying the time recording database for effort information about certain work packages or applying software measures on source code files. Thus we can provide Maven plugins for arbitrary software measures by combining these API calls respectively. Within the *Project Object Model*, the central XML configuration file of all Maven projects, the project manager can select which Maven plugin of our framework should be used and what parameters should be passed to it, in order to define the measurement scope.

Example Use of the Maven Measurement Framework

We have evaluated our approach on two software development projects that have been conducted by students within the bachelor's study courses in computer science at Bayreuth University. Two student teams had to implement a client for a *Mancala* game[6]. As the corresponding server containing the data for the actual state of the game was alread on-hand the students essentially had to implement the GUI and the application logic for the client. The latter also comprised the implementation of a game tree in order to find possible moves.

With our measurement framework we have monitored the development progress of the two teams (Figure 2). The size of the Java sources is counted in *Non Commenting Source Statements*[7]. It can be seen that team 1 at first has put its focus on the development of the GUI and has implemented the application logic not until the last four days. On the contrary, team 2 has equally developed both components in parallel from the beginning. This measurement based analysis has been verified by the resulting programs. It could be seen that the GUI of team 1 was more elaborate, whereas the program of team 2 was technically superior.

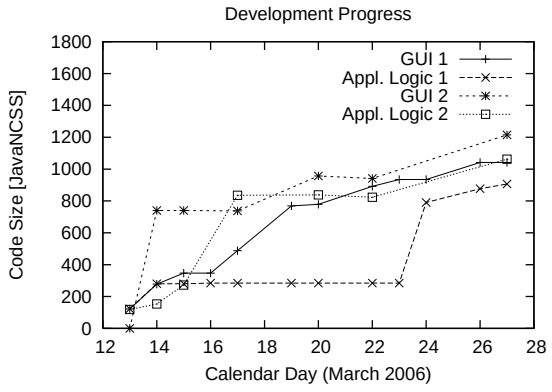

Figure 2: Comparison of the Implementation Progress of two Teams.

[6] http://en.wikipedia.org/wiki/Mancala

[7] http://www.kclee.de/clemens/java/javancss/

5 CONCLUSION

The automatic collection, interpretation and visualization of software measures is a complex task. The aim of this approach is to provide a lightweight tool for project managers to assess their development projects and the applied software process. The anchoring of software measures on elements of the process model enables the project manager to define software measures in advance before a concrete project starts. This allows for comparison of distribution ratios of effort and time spent on the activities of several projects in order to assess the progress of the current project or the productivity of the individual projects.

Therefore we think that with our approach to anchor software measures on elements of the process model and to identify by means of the work breakdown structure the entities that have to be measured we have found a practicable tradeoff between the automatic collection of information and the additional effort the developers have to perform.

The developers have to record their activities by indicating the associated WBS code within the time recording system. Furthermore these references to the work breakdown structure must also be maintained within the software configuration management and the bug tracking system. This is however a common modus operandi (Selby, 2005).

With our approach we gain the possibility to collect certain software measures in a standardized way that allows the cross-project comparison of the measurement results, because we define the entities to measure already before the start of the individual projects on the basis of the process model. The actual computation of the software measures is realized automatically at project runtime using Maven.

REFERENCES

Ambler, S. (2002). *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, New York.

Auer, M., Graser, B., and Biffl, S. (2003). A survey on the fitness of commercial software metric tools for service in heterogeneous environments: Common pitfalls. In *Proceedings of the Ninth International Software Metrics Symposium (METRICS 2003)*. IEEE Computer Society.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). Goal question metric paradigm. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons.

Basili, V. R. and Rombach, H. D. (1988). The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*, 14(6):758 – 773.

Beck, K. (2004). *Extreme Programming explained: Embrace Change*. Addison Wesley, Upper Saddle River, New Jersey, second edition.

Chrissis, M. B., Konrad, M., and Shrum, S. (2003). *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley, Boston.

Cockburn, A. (2001). *Agile Software Development*. Addison-Wesley, Boston, MA.

Dami, S., Estublier, J., and Amiour, M. (1998). APEL: A graphical yet executable formalism for process modeling. *Automated Software Engineering: An International Journal*, 5(1):61–96.

Futrell, R. T., Shafer, D. F., and Shafer, L. I. (2002). *Quality Software Project Management*. Prentice Hall, Upper Saddle River, NJ.

Johnson, P. (2001). You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering. In *The NSF Workshop for New Visions for Software Design and Productivity: Research and Applications*, Nashvile, TN.

Kempkens, R., Rösch, P., Scott, L., and Zettel, J. (2000). Instrumenting measurement programs with tools. In *PROFES '00: Proceedings of the Second International Conference on Product Focused Software Process Improvement*, pages 353–375, London. Springer.

Kruchten, P. (2003). *The Rational Unified Process - An Introduction*. Addison Wesley, Boston, third edition.

Lott, C. M. (1996). *Measurement-based Feedback in a Process-centered Software Engineering Environment*. PhD thesis, University of Maryland.

McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., and Hall, F. (2002). *Practical Software Measurement: Objective Information for Decision Makers*. Addison-Wesley, Boston.

Münch, J. and Heidrich, J. (2004). Software project control centers: Concepts and approaches. *Journal of Systems and Software*, 70:3–19. Issues 1–2.

Project Management Institute (2001). *Practice Standard for Work Breakdown Structures*. Project Management Institute.

Putnam, L. H. and Myers, W. (2003). *Five Core Metrics: The Intelligence behind successful Software Management*. Dorset House Publishing Co., New York.

Russac, J. (2002). Cheaper, better, faster: A measurement program that works. In International Function Point Users Group, editor, *IT Measurement: Practical Advice from the Experts*. Addison-Wesley, Boston, Massachusetts.

Selby, R. W. (2005). Measurement-driven dashboards enable leading indicators for requirements and design of large-scale systems. In *11th IEEE International Symposium on Software Metrics (METRICS 2005)*. IEEE Computer Society.