Zweitveröffentlichung

Daubner, Bernhard; Henrich, Andreas; Westfechtel, Bernhard

Integrierte Softwaremessung durch Verankerung der Softwaremaße an Elementen des Vorgehensmodells

Datum der Zweitveröffentlichung: 21.11.2023

Verlagsversion (Version of Record), Konferenzveröffentlichung

Persistenter Identifikator: urn:nbn:de:bvb:473-irb-919129

Erstveröffentlichung

Daubner, Bernhard; Henrich, Andreas; Westfechtel, Bernhard (2006): „Integrierte Softwaremessung durch Verankerung der Softwaremaße an Elementen des Vorgehensmodells“. In: Bettina Biel, Software Engineering 2006 : Fachtagung des GI-Fachbereichs Softwaretechnik, 28. - 31. März 2006 in Leipzig, Bonn: Gesellschaft für Informatik e. V., S. 157–162.

Rechtehinweis

Integrierte Softwaremessung durch Verankerung der Softwaremaße an Elementen des Vorgehensmodells

Bernhard Daubner[1] Andreas Henrich[2] Bernhard Westfechtel[1]

[1] Lehrstuhl für Angewandte Informatik I, Universität Bayreuth
bernhard.daubner@uni-bayreuth.de
bernhard.westfechtel@uni-bayreuth.de

[2] Lehrstuhl für Medieninformatik, Universität Bamberg
andreas.henrich@wiai.uni-bamberg.de

Abstract: Wird Softwaremessung mit dem Ziel der Prozessverbesserung angewendet, so sollte die Erfassung der Messwerte in geeigneter Weise standardisiert erfolgen, um verschiedene Softwareentwicklungsprojekte vergleichen zu können. Dieser Beitrag stellt einen Ansatz vor, die Softwaremaße an den Elementen des dem Entwicklungsprozess zugrunde liegenden Vorgehensmodells zu verankern.

Neu dabei ist, dass wir nicht nur Projektmeilensteine, sondern beliebige im Vorgehensmodell definierte Aktivitäten oder Produkte als Anknüpfungspunkte für die Softwaremaße verwenden. Dadurch ist es möglich, die relevanten Softwaremaße unabhängig von einem konkreten Projekt festzulegen. Über den Projektstrukturplan wird zur Laufzeit des Projekts die Verbindung von den Aktivitäten des Vorgehensmodells zu den tatsächlich zu messenden Entitäten hergestellt. Durch den Einsatz des Projektverwaltungstools *Maven* kann dann die Erfassung der Softwaremaße automatisiert durchgeführt werden.

1 Einführung und Motivation

Der grundsätzliche Nutzen der Softwaremessung ist in der Literatur unbestritten. Sie wird als Schlüssel für das Verständnis von Entwicklungsprozessen und Produkten betrachtet. Insbesondere wird sie als Voraussetzung für die Beurteilung der Softwarequalität und die Vorausberechenbarkeit von Softwareprojekten gesehen [Zus98].

Softwaremaße können im Rahmen des Managements von Softwareentwicklungen sowohl taktisch im Rahmen der Planung und Bewertung einzelner Softwareprojekte als auch strategisch zur Prozessverbesserung eingesetzt werden. Dabei macht es nach Meinung der Autoren Sinn, das Erheben der Softwaremaße zu standardisieren. Insbesondere sollten diejenigen Softwaremaße, die für das Projektmanagement von Interesse sind, jeweils zu vergleichbaren Zeitpunkten im Rahmen des Projektverlaufs erhoben werden. Dies führt zu einer Liste von Softwaremaßen, die grundsätzlich bei jedem durchzuführenden Projekt zur Anwendung kommen. Putnam und Myers stellen mit den Softwaremaßen *Produktumfang*, *Zeitbedarf*, *Aufwand*, *Qualität* und *Produktivität*, die sie als die *Five Core Metrics* [PM03] bezeichnen, eine derartige Liste vor.

2 Softwaremaße im Kontext von Vorgehensmodellen

2.1 Anknüpfungspunkte für Softwaremaße

Wir möchten daher im Folgenden einen Ansatz vorstellen, bei dem die durchzuführenden Softwaremessungen nicht im Bezug auf konkrete Produkte eines Softwareentwicklungsprojektes definiert, sondern statt dessen die Elemente des dem Projekt zugrunde liegenden Vorgehensmodells als Anknüpfungspunkte für die Softwaremaße verwendet werden. Dadurch können die zu ermittelnden Softwaremaße *ex ante* und unabhängig von einem konkreten Projekt definiert werden. Auch kann somit festgelegt werden, welche Softwaremaße grundsätzlich bei jedem durchzuführenden Softwareentwicklungsprojekt zur Anwendung kommen sollen, um Vergleiche zwischen den einzelnen Projekten durchzuführen.

Am Beispiel des *V-Modell-XT* [Bun04] soll diese Vorgehensweise näher erläutert werden. Bei diesem Vorgehensmodell dienen sogenannte Vorgehensbausteine als grundlegende, strukturierende Einheiten, welche jeweils eine Aufgabenstellung repräsentieren, die im Rahmen eines V-Modell-Projekts auftreten kann. Dabei werden diejenigen Produkte, Aktivitäten und Rollen in einem Vorgehensbaustein zusammengefasst, welche für die Erfüllung der zugehörigen Aufgabenstellung relevant sind. Verwandte Aktivitäten und Produkte, welche jeweils vorgehenstechnisch zusammengehören, werden dabei zu Aktivitätsgruppen bzw. Produktgruppen zusammengefasst. Die so gebildeten Vorgehensbausteine, die intern hierarchisch aufgebaut und die modularen Einheiten des V-Modells bilden, verwenden wir nun als Anknüpfungspunkte für die zu untersuchenden Softwaremaße.

Um beispielsweise festzustellen, wieviel Source-Code während eines Softwareentwicklungsprojekts im Rahmen der einzelnen Aktivitätsgruppen tatsächlich erstellt wird, schlagen wir vor, das entsprechende Softwaremaß für den Umfang des erstellten Source-Codes (z.B. LOC) direkt an den diesbezüglich interessanten Aktivitätsgruppen oder auch an den Vorgehensbausteinen zu verankern. Damit erhalten wir die Maße

- `LOC` im Rahmen der Aktivitätsgruppe *Software-Architektur* und
- `LOC` im Rahmen des Vorgehensbausteins *Anforderungsfestlegung*,

die darüber Auskunft geben, welcher Code-Anteil außerhalb der eigentlichen Programmieraktivitäten erzeugt wird.

2.2 Der Projektstrukturplan

Damit ergibt sich jedoch die Anforderung, diejenigen Artefakte zu identifizieren, die im Rahmen einer Aktivität aus dem Vorgehensmodell erstellt oder verändert wurden, um dann anschließend deren Code-Umfang messen zu können. Zu diesem Zweck verwenden wir den Projektstrukturplan als Bindeglied zwischen dem Vorgehensmodell, das dem Projekt zugrunde liegt, und den zu messenden Artefakten, die im Rahmen von Aktivitäten aus dem Vorgehensmodell erzeugt werden.

Der Projektstrukturplan (PSP) ist nach DIN 69901 (Projektmanagement) [Dt.87] die Darstellung der Gesamtheit der wesentlichen Beziehungen zwischen den Elementen eines Projektes. Dabei wird das Projekt hierarchisch in Teilaufgaben und Arbeitspakete strukturiert,

wobei sich eine Baumstruktur, wie in Abbildung 1 dargestellt, ergibt. Diese enthält alle Projektaktivitäten, die in den einzelnen Entwicklungsphasen durchzuführen sind.

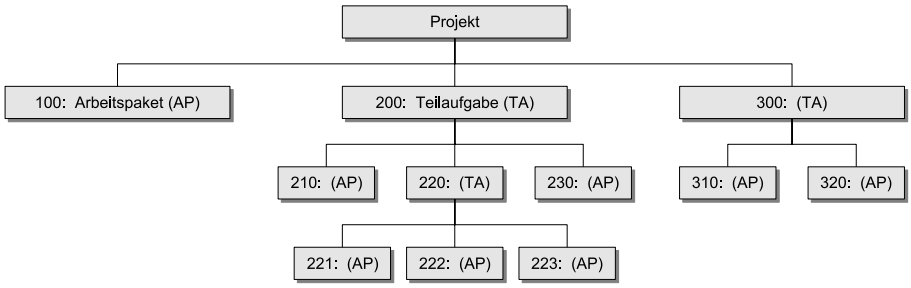

Abbildung 1: Zerlegung eines Projektes in Teilaufgaben und Arbeitspakete

2.3 Identifikation der zu messenden Entitäten über den PSP-Code

Wenn wir nun davon ausgehen, dass den Aktivitäten aus dem Vorgehensmodell entsprechende Arbeitspakete im Projektstrukturplan zugeordnet sind, so haben wir die oben genannte Anforderung, die zu einer Aktivität gehörenden Artefakte zu identifizieren, auf die Bestimmung der innerhalb eines bestimmten Arbeitspaketes erzeugten Artefakte reduziert. Dies ist jedoch mittels der identifizierenden Schlüssel (PSP-Codes) möglich, die üblicherweise jedem Strukturelement des Projektstrukturplans zugeordnet sind [GPM98]. Bei der *dekadischen Codierung* (vgl. Abbildung 1) gibt die Anzahl der Ziffern ungleich Null im PSP-Code den Hinweis auf die Einordnung des PSP-Elements innerhalb der hierarchischen Struktur des Projektstrukturplans.

Die PSP-Codes identifizieren jedoch nicht nur die Arbeitspakete des Projektstrukturplans, sondern können auch zur Kennzeichnung der innerhalb der jeweiligen Arbeitspakete erstellten oder veränderten Artefakte verwendet werden. Dazu ist es notwendig, dass die Artefakte in einem Versionsverwaltungssystem versioniert werden. Dann kann, immer wenn für eine veränderte Version eines Artefakts ein *Commit* im Versionsverwaltungssystem durchgeführt wird, mit der Commit-Information der PSP-Code des Arbeitspakets gespeichert werden.

Somit sind über die PSP-Codes in den Commit-Informationen des Versionsverwaltungssystems die Artefakte und auch die Veränderungen von Artefakten mit den entsprechendnen Aktivitäten im Vorgehensmodell verknüpft, so dass Softwaremaße, die an Elementen des Vorgehensmodells verankert wurden, entsprechend angewendet werden können.

Um die im Vorgehensmodell verankerten Softwaremaße über Projektgrenzen hinweg anwenden zu können, muss zunächst aus dem unternehmensspezifischen Prozessmodell ein Standard-Projektstrukturplan [Bur02] abgeleitet werden. Für die eigentlichen individuellen Softwareentwicklungsprojekte wird dieser Standard-Projektstrukturplan dann entsprechend ergänzt oder auch gekürzt. Diejenigen Softwaremaße, die mit den Elementen des unternehmensspezifischen Prozessmodells verknüpft sind, können dann über die projektübergreifend einheitlichen PSP-Codes der zugehörigen Teilaufgaben und Arbeitspakete erhoben werden.

3 Integration der Softwaremessung in den Entwicklungsprozess

3.1 Maven als Projektverwaltungstool

Da für die Integration und automatische Berechnung der im vorhergehenden Abschnitt beschriebenen Softwaremaße eine entsprechende Werkzeugunterstützung nötig ist, wollen wir uns an dieser Stelle auf Java-Projekte beschränken. Zur Implementierung unseres Ansatzes verwenden wir die Software *Maven* aus dem Apache-Projekt [Apa05b]. Maven ist ein Verwaltungs- und Entwicklungsunterstützungstool für Java-Projekte. Zum einen unterstützt es den Entwickler beim sogenannten *Build-Process*, d.h. es kompiliert den Source-Code und berücksichtigt dabei Abhängigkeiten wie zusätzlich benötigte Bibliotheken. In dieser Hinsicht kann Maven als Alternative zu *Ant* [Apa05a] betrachtet werden.

Darüber hinaus stellt es die Möglichkeit zur Verfügung, wichtige Informationen über Projektmitarbeiter, Projektbeteiligte, Projektressourcen (z.B. Dokumente, Lizenzen) und das innerhalb des Projekts verwendete Versionverwaltungs- bzw. Bug-Tracking-System zu verwalten. Durch Plugins kann die Funktionalität von Maven jederzeit erweitert werden. Die Plugins können dabei auf die Projektinformationen zurückgreifen und dadurch z.B. generisch auf das verwendete Versionsverwaltungssystem zugreifen, ohne den Typ des Versionsverwaltungssystems genau kennen zu müssen.

3.2 Messwertgeber für Softwaremaße

An dieser Stelle setzen wir nun an und verwenden Maven-Plugins, um sogenannte *Messwertgeber* zu implementieren. Darunter verstehen wir ein Maven-Plugin, welches ein bestimmtes Softwaremaß berechnen kann. Beispielsweise kann der `JavaLOC`-Messwertgeber die *Lines of Code* von Java-Programmdateien bestimmen. Darüber hinaus kann ein Messwertgeber so konfiguriert werden, dass er sein Softwaremaß nur im Kontext eines bestimmten Elements des Vorgehensmodells ermittelt.

Angewandt auf das Beispiel aus Abschnitt 2.1 kann der `JavaLOC`-Messwertgeber durch entsprechende Konfiguration an der Aktivitätsgruppe *Software-Architektur* des Vorgehensmodells verankert werden. Aus dem Projektstrukturplan ist bekannt, welche Arbeitspakete zu dieser Aktivitätsgruppe gehören. Durch Abfrage des Versionsverwaltungssystems, welches ebenfalls in der Maven-Projekt-Konfiguration referenziert wird, können anhand der Commit-Informationen die im Rahmen dieser Arbeitspakete erzeugten oder veränderten Source-Dateien mit den zugehörigen Revisionsnummern identifiziert werden. Anschließend muss durch sukzessives *Check-Out* der betroffenen Versionen dieser Source-Dateien deren Umfangsänderung hinsichtlich der LOC im Rahmen dieser Aktivitätsgruppe ermittelt werden.

Darüber hinaus ist es möglich, mehrere Plugins (und damit Messwertgeber) zu kombinieren. Die Tätigkeiten, welche ein bestimmtes Maven-Plugin ausführen kann, werden als *Goals* bezeichnet, wobei jedes Plugin mindestens ein Goal hat. Da hinter diesen Goals letztendlich API-Funktionen des jeweiligen Plugins stehen, können diese auch von anderen Plugins aus aufgerufen und die Ergebnisse miteinander verknüpft werden. So kann ein `JavaLOC`-Messwertgeber mit einem `Effort`-Messwertgeber kombiniert werden, wodurch man einen `Productivity`-Messwertgeber erhält. Dieser ruft zunächst den

`JavaLOC`-Messwertgeber auf, um den Code-Umfang zu bestimmen. Anschließend wird der `Effort`-Messwertgeber aufgerufen, der durch Abfragen der Zeiterfassungsdatenbank den Zeitaufwand bestimmt. Dazu sind auch bei der Zeiterfassung die jeweils zugehörigen Arbeitspakete mit anzugeben. Die Messwerte beziehen sich dabei jeweils auf die gleichen Arbeitspakete. Schließlich kann dann die Produktivität bei der Bearbeitung dieser Arbeitspakete berechnet werden.

4 Anwendungsgebiete, verwandte Ansätze und Zusammenfassung

Fenton und Pfleeger [FP98] weisen bezüglich der Durchführung von Softwaremessungen darauf hin, dass die Datenerfassung hinreichend einfach sein muss, um den normalen Arbeitsablauf möglichst wenig zu stören, und dass die Daten letztendlich in einer Datenbank abgelegt werden müssen.

Noch einen Schritt weiter geht Johnson in seinem Konferenzbeitrag „You can't even ask them to push a button: [...]" [Joh01], in dem er den Ansatz des Tools *Hackystat* vorstellt. Die Hackystat-Software erfasst über Sensoren die Tätigkeiten des einzelnen Anwenders innerhalb der Softwareentwicklungsumgebung und analysiert und protokolliert somit, welche Arbeiten (z.B. Programmierung, Fehlersuche, Modellierung) der Anwender gerade durchführt, wobei dieser in seiner Tätigkeit in keiner Weise unterbrochen wird.

Wir stimmen mit Johnson überein in der These, dass Softwareentwickler nur schwer dazu zu bewegen sind, neben ihrer eigentlichen Programmiertätigkeit auch noch Verwaltungsinformationen zu dokumentieren und zu protokollieren. Aber alle relevanten Informationen über den Entwicklungsprozess wird man kaum vollständig automatisiert erfassen können. Wir sind der Meinung, mit unserem Ansatz, die Softwaremaße an Elementen des Prozessmodells zu verankern und mittels des Projektstrukturplans die zu messenden Entitäten zu identifizieren, einen praktikablen Kompromiss zwischen der Informationsgewinnung und dem Mehraufwand für die Anwender entwickelt zu haben.

Unser Ansatz ist neu, da wir nicht nur wie beispielsweise in [Boe96] Projektmeilensteine, sondern beliebige im Vorgehensmodell definierte Aktivitäten oder Produkte als Anknüpfungspunkte für die Softwaremaße verwenden. Darüber hinaus sieht unser Ansatz vor, dass die zur Projektlaufzeit zu messenden Entitäten automatisch bestimmt werden, um die vorher festgelegten Softwaremaße darauf anwenden zu können.

Der Mehraufwand für den Anwender besteht zum einen darin, seine Tätigkeiten unter Angabe der Arbeitspaketnummern der Projekte im Zeiterfassungssystem zu protokollieren. Diese Vorgehensweise ist in der Industrie aber nicht unüblich, was durch das Vorhandensein entsprechender Software belegt wird [SAP03]. Zusätzlich müssen bei der Speicherung von Daten im Versionsverwaltungs- und im Bug-Tracking-System ebenfalls entsprechende Daten gepflegt werden. Auch dies ist eine übliche Vorgehensweise [Sel05]. Insbesondere können einige Versionsverwaltungssysteme so konfiguriert werden, dass für den *Check-in* zwingend entsprechende *Change-IDs* (also z.B. PSP-Codes) angegeben werden müssen. Diese PSP-Codes könnte man sogar automatisch hinterlegen, wenn ein Prozessplanungssystem verwendet wird, in dem die Schritte des Vorgehensmodells explizit modelliert und den Entwicklern als Aufgaben zugeordnet werden. Als Beispiel für ein

derartiges System sei das *AHEAD*-System (ein Forschungs-Prototyp) [HSW04] genannt, das von einem der Autoren entwickelt wurde.

Derzeit haben wir unter Verwendung von Maven unseren Ansatz nur sehr prototypisch implementiert, um das grundsätzliche Funktionieren unserer Vorgehensweise zu überprüfen. Insbesondere erfolgt derzeit die Erstellung der Liste mit den von einem Arbeitspaket betroffenen Dateien und deren *Check-out* aus dem Versionsverwaltungssystem manuell. Unsere nächsten Schritte werden daher sein, hier entsprechende Plugins zu entwickeln. Mittelfristig möchten wir diesen Ansatz in einem entsprechenden Softwareentwicklungsprojekt, evtl. mit einem externen Kooperationspartner, evaluieren.

Literatur

[Apa05a] Apache Software Foundation. Apache Ant Project, 2005. http://ant.apache.org.

[Apa05b] Apache Software Foundation. Apache Maven Project, 2005. http://maven.apache.org.

[Boe96] Barry Boehm. Anchoring the Software Process. *IEEE Softw.*, 13(4):73–82, 1996.

[Bun04] Bundesministerium des Innern (BMI). V-Modell XT, Version 1.01, 2004. http://www.v-modell-xt.de.

[Bur02] Manfred Burghardt. *Projektmanagement*. Publicis Corporate Publishing, Erlangen, sixth edition, 2002.

[Dt.87] Dt. Institut für Normung. *DIN 69901 - Projektwirtschaft: Projektmanagement*, 1987.

[FP98] Norman E. Fenton und Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company, Boston, Massachusetts, second edition, 1998.

[GPM98] GPM Deutsche Gesellschaft für Projektmanagement e.V. *Projektmanagement-Fachmann*, Band 2. RKW-Verlag, Eschborn, fourth edition, 1998.

[HSW04] Markus Heller, Ansgar Schleicher und Bernhard Westfechtel. Process Evolution Support in the AHEAD System. In John L. Pfaltz, Manfred Nagl und Boris Böhlen, Hrsg., *Applications of Graph Transformations with Industrial Relevance: Second International Workshop, AGTIVE 2003*, Band 3062 of *Lecture Notes in Computer Science*, Seiten 454 – 460, 2004.

[Joh01] Philip Johnson. 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, December 2001.

[PM03] Lawrence H. Putnam und Ware Myers. *Five Core Metrics: The Intelligence behind successful Software Management*. Dorset House Publishing Co., New York, 2003.

[SAP03] SAP AG. *Arbeitszeiterfassung mit SAP CATS*. Walldorf, 2003.

[Sel05] Richard W. Selby. Measurement-Driven Dashboards Enable Leading Indicators for Requirements and Design of Large-Scale Systems. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 2005.

[Zus98] Horst Zuse. *A Framework of Software Measurement*. de Gruyter, Berlin, New York, 1998.