1 Einleitung

Inhaltsangabe

Seit Beginn der Softwarekrise Mitte der 1960er Jahre stellt sich immer wieder die Frage: Wie stellt man sicher, dass ein Softwareprojekt erfolgreich durchgeführt wird[Rum10]? Oder, etwas schwächer formuliert in Anerkennung der empirischen Erkenntnis, dass der Stein der Weisen noch nicht gefunden wurde und auch und gerade heute Softwareprojekte weiterhin scheitern [Cha05]: Wie erhöht man zumindest die Erfolgs-Wahrscheinlichkeit? Die Komplexität von Software steigt immer weiter und damit das Risiko des Scheiterns. Zahlreiche Forschungsprojekte haben das Ziel, dieses Risiko zu mindern. Softwareentwicklungs-Prozesse wie etwa das Spiral Modell, der Rational Unified Process, Extreme Programming[LL10] und Scrum[1] strukturieren den Vorgang der Softwareentwicklung. Sie definieren Rollen, die Mitarbeiter einnehmen, sowie Aktivitäten und Artefakte (Dokumente), die diese durchzuführen bzw. zu erstellen haben. Auf diese Weise wird das komplexe Software-Projekt in handhabbarere, kleinere Teile zerlegt. Prozessverbesserungs-Modelle wie CMMI oder ISO9000 bewerten Organisationen daraufhin, wie gut sie ihren definierten Prozessen folgen[2].

Sowohl Softwareentwicklungs-Prozesse als auch Prozessverbesserungs-Modelle bedienen sich Metriken, um Aussagen über Qualitätseigenschaften der zugrundeliegenden Entität zu treffen. Erstere nutzen Metriken wie etwa Lines of Code, zyklomatische Komplexität oder Testabdeckung, um Eigenschaften von Software zu bewerten. Das Prozessverbesserungs-Modell CMMI nutzt sogenannte SCAMPI-Untersuchungen, in denen festgestellt wird, in welchem Reifegrad sich die in der Organisation eingesetzten Prozesse befinden. Während die Metriken für Software bereits ausgiebig erforscht sind und mit Werkzeugunterstützung (z.B. Sonar[3]) versehen sind, ist insbesondere Letzteres für Metriken auf Softwareentwicklungs-Prozessen Mangelware.

1.1 Ziel der Arbeit

Im Rahmen dieser Diplomarbeit soll herausgefunden werden, ob und inwiefern die Datenerhebung und Darstellung von Metriken für Softwareentwicklungs-Prozesse automatisiert werden kann und inwiefern diese Metriken hilfreiche Erkenntnisse über die Umsetzung des Softwareentwicklungs-Prozesses liefern.

[1] `http://www.scrumalliance.org/`

[2] Übersetzt von `http://en.wikipedia.org/wiki/Software_development_process`: "Independent assessments grade organizations on how well they follow their defined processes"

[3] `http://www.sonarsource.org/`

"You can't even ask them to push a button" (Man kann sie nicht einmal darum bitten, einen Knopf zu drücken)[Joh01]: Mit diesem prägnanten Titel stellt Johnson die Abneigung von (nicht nur) Entwicklern heraus, zusätzliche Aktivitäten wie etwa Ist- und Soll-Aufwand-Erfassung durchzuführen, die nur "für die Qualitätssicherung" gut sind und keinen direkten Nutzen für den Entwickler zu haben scheinen. Er fand heraus (auch aus eigener Erfahrung), dass selbst nur einen Knopf zu drücken unzumutbar sei ("too much overhead"). Dieses Hindernis wird durch nicht-störende ("non-disruptive") Werkzeuge, d.h. Werkzeuge, die ohne aktive Interaktion mit dem Entwickler funktionieren, aus dem Weg geräumt.

Softwareentwicklungs-Prozesse, so die Idee dieser Arbeit, spiegeln sich in den Werkzeugen wider, die von den Entwicklern ohnehin genutzt werden, wie etwa Change-Request-Systeme. Sieht etwa ein agiler Prozess wie Scrum vor, dass User Stories (und daraus resultierende Aufgaben) innerhalb eines Sprints zu erledigen sind, so kann man diese Aufgaben und ihren Fortschritt ("ausstehend", "in Bearbeitung", "erledigt") im Change-Request-System wiederfinden.

Kann man die Daten, die in diesen Werkzeugen entstehen, hinsichtlich einer Bewertung der Umsetzung des Prozesses sinnvoll auswerten? Die Change-Request-Systeme selbst bieten dafür wenig Unterstützung. Sie richten sich vor allem an ihre direkten Nutzer, die Entwickler. Daher soll während der Diplomarbeit ein Werkzeug entwickelt werden, das diese Informationen aufbereitet. In Evaluationen des Werkzeugs mit Projektleitern soll anschließend ermittelt werden, welche Daten und welche Darstellungen als sinnvoll und hilfreich erachtet werden. Erkennt man Probleme im Softwareentwicklungs-Prozess leichter? Welche Informationen aus den Change-Request-Systemen sind interessant und welche eher nicht?