Joachim Marwinski

(joachim.marwinski@pixsoftware.de)
ist seit 2005 IT-Berater und Projektleiter bei der Pix Software GmbH. Seit 16 Jahren beschäftigt er sich mit dem Thema Softwareentwicklung aus verschiedenen Perspektiven.
Als B.Sc. Informatik mit dem Schwerpunkt Software Engineering interessiert er sich insbesondere für die Verbesserung des Softwareentwicklungsprozesses. Bei der Pix Software GmbH steht daher die Beratung zu Softwareprodukten rund um den Softwareentwicklungsprozess (JIRA, Confluence, Contour, Mingle, Bamboo) im Fokus seiner Arbeit.

Agile braucht Organisation

Gerade die agile Softwareentwicklung braucht die Organisation der Aufgaben und Dokumente, damit die Beteiligten für die eigentliche Arbeit die Hände frei haben. Dabei helfen Softwarewerkzeuge von der Anforderungserfassung bis zum abschließenden analytischen Bericht der im Projekt gesammelten Metriken. Wie greifen Werkzeuge ineinander, wie werden bestehende Systeme integriert, und in welcher Kombination sollten Werkzeuge vorhanden sein? Die hier beschriebenen Zusammenhänge gelten für unterschiedliche Vorgehensmodelle. Daher wird im Folgenden jeweils eines dieser Vorgehensmodelle herangezogen, wenn es um eine bestimmte Ausprägung eines Prinzips geht. Insbesondere wird in diesem Artikel die Verwaltung von Dokumenten und Aufgaben mit den Werkzeugen Contour bzw. Mingle und JIRA beleuchtet.

Dokumente

Unabhängig vom gewählten Vorgehensmodell entstehen in allen Projekten neben dem Quellcode viele weitere Dokumente (oft auch allgemeiner als Artefakte bezeichnet). Die Anzahl der Dokumente ist in den schwergewichtigen Modellen eher sehr hoch, in den agilen Modellen ist die Reduktion der Anzahl der Dokumente ein wichtiger Aspekt der Modelle. Trotzdem entstehen im Laufe eines agilen Projektes ‚viele' Dokumente. Diese Dokumente enthalten z.B. Anforderungsbeschreibungen, Spezifikationen, Modelle, Testbeschreibungen oder stellen weitere Dokumenttypen dar. Je nach Vorgehensmodell gibt es verschiedene Bezeichnungen für die Dokumenttypen. Eine Anforderung heißt unter XP beispielsweise „Story", unter Scrum ist sie das „product backlog item". Sie hat dort einen anderen Umfang und wird flexibler geändert. Gleichzeitig hat sie eine andere Aufgabe als z.B. im RUP Rational Unified Process. Wichtig bei der Betrachtung im Rahmen dieses Artikels ist festzuhalten, dass Anforderungen beschreibende Dokumente in allen Modellen entstehen, auch wenn die Anforderungsdokumente in den Modellen verschieden aufgebaut sind und in anderer Art und Weise verwendet werden. Ein „product backlog item" ist in diesem Sinne ebenfalls ein Anforderungsdokument, auch wenn es lediglich als Einzeiler in einer Tabelle eingetragen wird.

Die agilen Vorgehensmodelle wie XP, Scrum, FDD etc. versuchen mit möglichst wenigen Dokumenten auszukommen und genaue, detaillierte Spezifikationen durch leichtgewichtigere Anforderungsdokumente (z. B. „product backlog items") zu ersetzen. Trotzdem entstehen zahlreiche Dokumente bzw. „items", und diese müssen zentral gespeichert und verwaltet werden, um eine effiziente Zusammenarbeit zu ermöglichen. Die agilen Methoden schreiben hierzu keine Methodik vor, es können also z. B. Anforderungsbeschreibungen oder „product backlog items" als Word-Dokumente auf einem Netzlaufwerk im Dateisystem gespeichert werden. Dadurch entstehen allerdings einige Nachteile: Die Dokumente sind nicht versioniert (oft werden sie durch das Verschicken in Emails "versioniert"), die Dokumente haben ggf. einen uneinheitlichen Aufbau, es gibt keine strukturierten Informationen, die etwa aus Ausklapplisten abgerufen werden könnten wie z. B. einer Statusliste oder einer Komponentenliste mit Elementen wie „Webschnittstelle", usw..

Strukturierte Informationen ermöglichen eine Auswertung in Berichten (z.B. „erledigte versus offene Aufgaben"/ in der rückblickenden Projektbetrachtung) oder vereinfachen das Erstellen von Dokumentlisten einer bestimmten Kategorie. Das ist mit immer unterschiedlichen Freitexteingaben wie „Web-Schnittstelle", „Web Schnittstelle", „Web-Interface" nicht möglich.

Werden die Dokumente statt in einer Verzeichnisstruktur in einem verwaltenden System gespeichert und dort nach Dokumenttyp in einer hierarchischen Struktur abgelegt, dann ergeben sich daraus viele Vorteile: Die Dokumente sind in einzelnen Projekten an einer zentralen Stelle zu finden, der Zugriff kann leichter berechtigt werden. Der Zugriff für Kunden oder externe Dienstleister lässt sich wesentlich einfacher realisieren, wenn das verwaltende System über einen Webbrowser verwendet werden kann. Es ist dann kein Zugang auf die internen Laufwerke erforderlich. Arbeiten in verteilten Teams wird so wesentlich erleichtert.

Beziehungen zwischen Dokumenten

Viele Dokumente und Artefakte stehen zu anderen in einem Zusammenhang. Eine Testspezifikation beispielsweise ist abhängig von der Definition der Anforderung. Ändert sich die Anforderung, so muss auch die Testfallspezifikation geändert werden, zumindest muss das Erfordernis der Anpassung auf eine einfache Art überprüft werden können. Dies trifft natürlich auf alle Vorge-

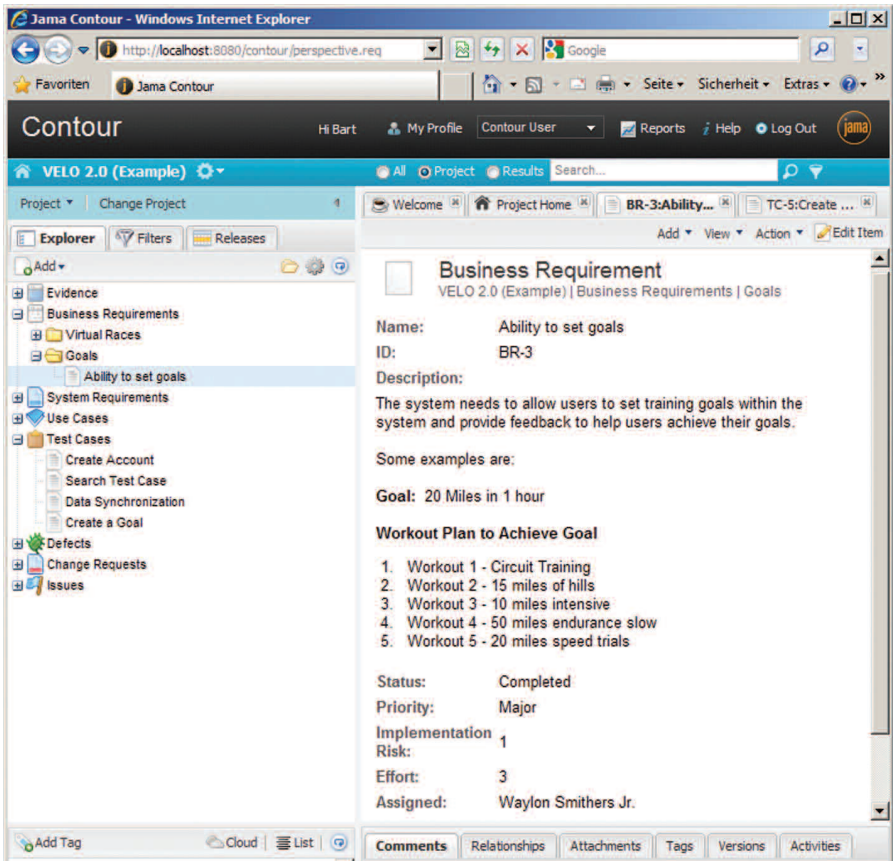

Abbildung 1: Nach Objekttyp in einer Ordnerhierarchie gespeicherte Dokumente

hensmodelle gleichermaßen zu, egal, wie spezifikationsreich ein Vorgehensmodell ist. Hier kann die Verknüpfung von Dokumenten bereits ermöglichen, die zu prüfenden Dokumente zu finden. Komfortabler ist es, wenn eine Analyse der Änderungszeitpunkte der Dokumente ermöglicht, die Menge der zu prüfenden Dokumente einzuschränken. Die inhaltliche Prüfung der Änderungen und ihrer Auswirkungen auf das abhängige Dokument bleibt dabei weiterhin dem Benutzer überlassen- ein Automatismus ist i.d.R. nicht möglich. Wird testgetrieben entwickelt, so wird möglicherweise anstelle einer Testspezifikation für die automatisierten Tests direkt der Testquelltext als Testspezifikation verwendet. Dann bleibt der beschriebene Zusammenhang weiterhin für die nicht automatisierbaren Tests gültig.

Eine weitere Abhängigkeit von Dokumenten/Artefakten liegt z.B. zwischen der Anforderung oder einer Fehlermeldung und dem Quellcode. Oft ist es zur Umsetzung einer geänderten oder einer ähnlichen Anforderung interessant, die geänderten Quellcodedateien zur ursprünglichen Anforderung zu finden. Hier kommt das Versionierungssystem ins Spiel. XP (und auch weitere Vorgehensmethoden) setzen die Verwendung von Versionierungssystemen wie z. B. CVS oder Subversion voraus, es sollte grundsätzlich immer eines vorhanden sein.

Werden die zur Lösung eines Fehlers erforderlichen Quellcodeänderungen in das Versionierungssystem übertragen, so können z. B. durch Angabe eines Fehlerkürzels in den commit-Kommentar (hier: TST-2) die Informationen zu den geänderten Dateien an das Fehlerverwaltungssystem weiter gegeben werden (siehe Abb. 2).

Aus Anforderungen werden Aufgaben

Um z. B. die Anforderungen für den Scrum Sprint zusammentragen zu können, ist bei nicht mehr „kleinen“ Projekten – dort mag eine Excel-Liste ausreichen – die Unterstützung durch ein Softwarewerkzeug sinnvoll. Dort kann das „product backlog“, die Liste der Anforderungen an die zu erstellende Software, abgerufen und durch den Inhaber der Rolle „product owner“ priorisiert werden. Über die Zuordnung zu einem Sprint – sein Ergebnis ist das „product increment“, eine dem Kunden vorzulegende Zwischenversion des Produktes – entsteht das „sprint backlog“. Die Teammitglieder entscheiden sich im Laufe des Sprints für die Erledigung der Aufgaben in der von ihnen ausgewählten Reihenfolge. In anderen Prozessen werden ebenfalls Aufgaben aus einer Menge an offenen Aufgaben zur Bearbeitung *angenommen*, während z. B. im RUP die Aufgaben *zugewiesen* werden. Wichtig ist, dass das verwendete Softwarewerkzeug die Beteiligten so gut unterstützt, dass das Einplanen für eine Version oder für einen Sprint und das Finden und Zuweisen (bzw. sich selbst zuweisen) von Aufgaben sehr einfach möglich ist. Nur dann können sich die Beteiligten auf ihre Arbeit konzentrieren. In folgender Abbildung ist eine Kartenansicht von Aufgaben zu sehen, die den bekannten Planungskärtchen entspricht. Bei den Planungskärtchen handelt es sich entweder direkt um Aufgaben, oder z. B. um „product backlog items“, denen Aufgaben fest zugeordnet sind. Die Kärtchen können in der Planungssicht durch Ziehen & Ablegen einer geplanten Version zugeordnet, oder durch Ziehen & Ablegen auf der Auf-

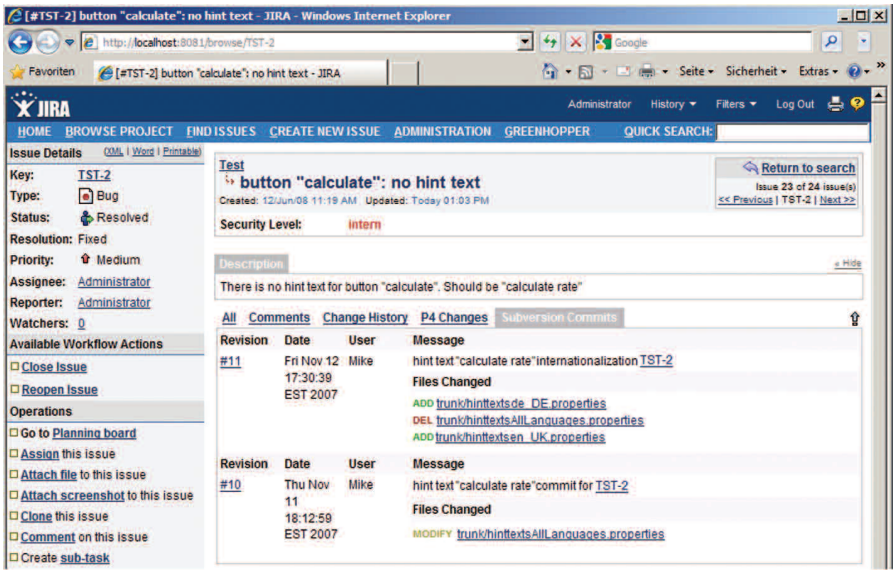

Abbildung 2: Anbindung Versionierungssystem

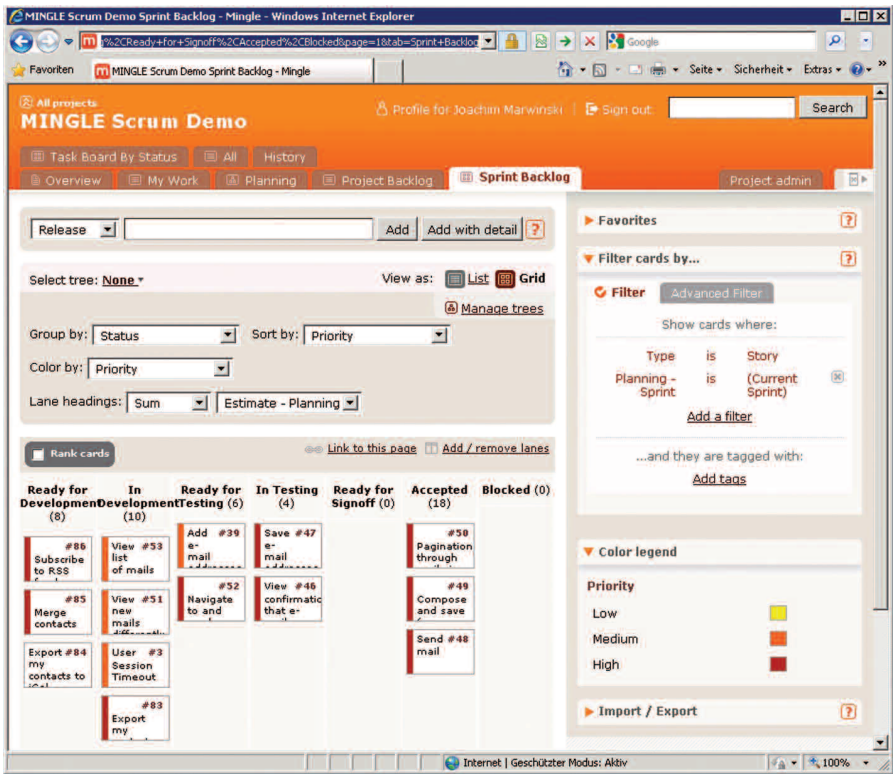

Abbildung 3: Liste der anstehenden Aufgaben in MINGLE: Sprint backlog

gabenseite in Arbeit genommen werden. So wird der Status eines Sprints oder einer Version schnell sichtbar: welche Aufgaben sind bereits erledigt, was steht noch an.

Aufgabenorientiertes Arbeiten

Die Eclipse-IDE-Erweiterung Mylyn vereinfacht das aufgabenorientierte Arbeiten für den Entwickler. Er muss mit Mylyn seine IDE nicht mehr verlassen und in das Aufgabenmanagementsystem wechseln, um sich eine Liste der anstehenden Aufgaben zu beschaffen. Die anstehenden Aufgaben werden innerhalb der IDE aufgelistet und können dort auch geändert werden, z. B. im Status von „offen“ auf „in Arbeit“ oder „erledigt“. Zusätzlich kann der Entwickler zu einer Aufgabe eine Menge von Dateien für eine Sicht definieren, an denen er zur Erledigung der Aufgabe arbeitet. Die anderen Dateien des Eclipse Projektes werden dann ausgeblendet, wodurch die Übersicht verbessert wird.

Abschätzungen

Wie kann ein Team bzw. ein Teamleiter (oder “product owner“) abschätzen, welche Änderungen innerhalb eines Sprints möglich sind, oder wann ein Release veröffentlicht werden kann?

Indem sie kontinuierlich ihre Schätzungen verbessern. Dazu wird zu irgendeinem Zeitpunkt mit dem Schätzen von Aufwänden begonnen. Wird eine Aufgabe abgeschlossen, so kann die ursprüngliche Schätzung mit dem tatsächlich benötigen Aufwand verglichen werden. Im Laufe der Zeit findet eine Verbesserung der Abschätzungen statt. Der Entwickler „lernt“ besser zu schätzen, und kann sich bei neuen Aufgaben an alten Schätzungen orientieren, da die Schätzungen schnell zugreifbar sind. Folgendes Bild zeigt eine Schätzung des Gesamtaufwandes für die Erledigung der Aufgabe „calculation of rating“: Eine ursprüngliche Schätzung von 1 Tag Gesamtaufwand, von dem bereits 2 Stunden geleistet wurden.

Auswertungen

Berichte z. B. aus Excel-Listen zusammenzutragen ist eine mühselige Aufgabe, die in unterschiedlichen Projekten immer wieder „von Hand“ durchgeführt werden muss. Wird ein Werkzeug zur Verwaltung von Aufgaben verwendet, so können aus den im System vorliegenden Aufgaben Berichte erstellt werden, die bestimmte Metriken auswerten, die während der Arbeit mit den Aufgaben automatisch erfasst wurden. Ein Beispiel hierfür ist ein Bericht „erstellte versus erledigte Aufgaben pro Zeiteinheit“. Durch solche Berichte kann einerseits der aktuelle Stand ermittelt werden, andererseits können nach Abschluss des Projektes die rückblickenden Auswertungen vereinfacht werden. Auch lässt sich über Funktionen wie Changelog / Roadmap schnell abfragen, welche Aufgaben (und damit auch: welche Anforderungen) in einer herausgegebenen Version des Produktes umgesetzt wurden bzw. für eine zukünftige Version geplant sind.

Wie greifen Werkzeuge ineinander?

Gut zusammenarbeitende Werkzeuge ermöglichen eine Übersicht auf Anforderungsebene und machen den Weg von der Anforderung über die Aufgaben zur geänderten Quellcodedatei deutlich.

Werden unterschiedliche Werkzeuge zur Verwaltung von Dokumenten und Aufgaben eingesetzt, so sollten diese Möglichkeiten bieten, die Daten über einen „Connector“ zu synchronisieren. Ein angeschlossenes Versionierungssystem erlaubt das Nachverfolgen von Anforderungen und Änderungswünschen bis hinunter auf die Quellcodeebene.

Integrierende Werkzeuge zum Fehler-/Aufgabenmanagement zeigen z. B. den Status eines Builds auf der Anwendungsstartseite in konfigurierbaren Fensterelementen an und ermöglichen so einen schnellen Überblick.

Fazit

In allen Softwareentwicklungsprojekten entstehen Dokumente (Artefakte, „items“,…) und Aufgaben, deren Verwaltung durch ein Werkzeug die alltägliche Arbeit erheblich vereinfacht. ■

Besuchen Sie uns auf unserer Website:
www.pixsoftware.de/einleitungen/OSO112009.html

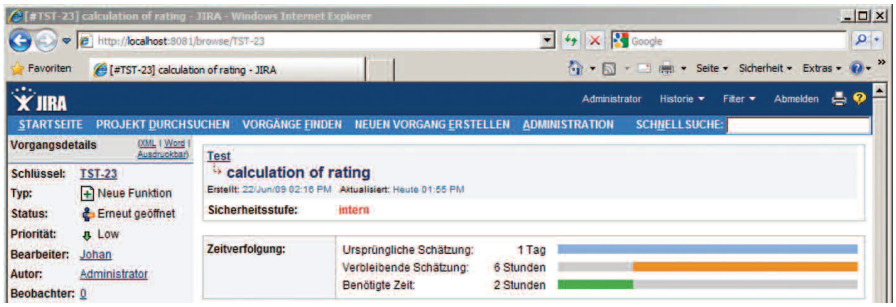

Abbildung 4: Aufwandsschätzungen