Einsatz Agiler Methoden und Techniken für sehr große Projekte

Martin Grundmann, Erwin Thurner

Nokia Siemens Networks, München
martin.grundmann@nsn.com

Abstract: Die agilen Vorgehensmodelle wie SCRUM [Sch04] und eXtreme Programming (XP) [WRL05] können in vielen Fällen die Effizienz in der Softwareentwicklung sowie die Zufriedenheit der Kunden und Mitarbeiter verbessern. Die Werte und Prinzipien dieser Vorgehensmodelle lassen sich aber in vielen Fällen nicht einfach "schematisch" auf die besonderen Anforderungen der Entwicklung großer Produktfamilien übertragen. Dieses Papier beschreibt unsere Anpassungen für den konkreten Einsatz agiler Techniken für die sehr große Softwareentwicklung des Telekommunikationssystems IMS von Nokia Siemens Networks. Der Fokus liegt hier auf den Phasen Analyse und Design, sowie einer detaillierteren Behandlung der Entwicklungsdokumentation.

1 Einleitung

Das „Manifest für Agile Softwareentwicklung“[1] proklamiert vier Werte, auf die sich ein Entwicklungsvorgehen fokussieren soll, um effizient kundengerechte Software zu entwickeln. Einer dieser Werte ist z.B. die *regelmäßige Freigabe von nutzbaren Zwischenständen der produzierten Software*, was als wertvoller erachtet wird als eine umfangreiche Dokumentation.

In den mehr als 10 Jahren seit dem Manifest wurden bereits vielfältige Erfahrungen mit agilen Vorgehensmodellen in konkreten Projekten gesammelt. Diese Erfahrungen sind überwiegend positiv, zeigen allerdings auch einige, ausreichend zu berücksichtigende Risiken. Große Unternehmen stehen hierbei oft vor dem Problem, dass sich aufgrund der gewachsenen Strukturen einige agile Prinzipien nur suboptimal umsetzen lassen, und neben den agilen Werten auch weitere Werte im Vordergrund stehen müssen [Boe03, LV10]. Zum Beispiel ist offshore-Entwicklung heutzutage gängige Praxis für große Entwicklungen, was der Kommunikation und Interaktion zwischen den Entwicklern Grenzen setzt, wodurch wiederum die Dokumentation als Mittel der Kommunikation an Wert gewinnt.

[1] http://agilemanifesto.org/iso/de

Nokia Siemens Networks führt seit der Gründung 2007 schrittweise agile Softwareentwicklung ein. Die Einführung agiler Methoden und Techniken für die Softwareentwicklung der Produktlinie IMS (IP-basiertes Multimedia-Subsystem) ist Thema dieses Papiers. Zunächst geben wir einen kurzen Überblick über den geschäftlichen, organisatorischen und technischen Kontext der IMS Produktlinie. Nach Vorstellung von Motivation und Randbedingungen für die Einführung agiler Methoden und Techniken, zeigen wir den gewählten agilen Entwicklungsprozess inkl. der Dokumentationsstruktur, mit dem Schwerpunkt Analyse und Design. Im Abschluss fassen wir die Erfahrungen zusammen.

2 Was ist IMS?

Das IP-Multimedia Subsystem (IMS) wird überwiegend von den internationalen Telekommunikations-Gremien ETSI/3GPP und ITU standardisiert als ein zentraler Bestandteil des Next Generation Networks (NGN). Das NGN stellt ein konvergentes Netzwerk dar, welches Endkunden eine Vielzahl von Diensten – wie Festnetz- und Mobiltelefonie, Internet und TV/Video – mittels einer einheitlichen Datenpaket-vermittelten Kommunikation über beliebige Zugangsnetze bereitstellt – im Gegensatz zu den klassischen, separaten, Leitungs-vermittelten Dienstnetzen. Im NGN ist das IMS die zentrale Instanz für die *Dienstvermittlung*, d.h. hier wird der Aufbau einer logischen Verbindung signalisiert, die vereinbarte Verbindung im Netz eingerichtet, und die dazugehörige Abrechnungsinformation generiert.

Stark vereinfachtet betrachtet besteht der Kern des IMS aus den zwei funktionalen Subsystemen *Call Session Control Function* (CSCF) und *Home Subscriber Server* (HSS). Der CSCF stellt die Funktionen für die Dienstvermittlung, während der HSS die Teilnehmerdaten verwaltet sowie die Authentisierung und Authorisierung der Teilnehmer sicherstellt. Ein IMS-Netz muss genauso verfügbar und ausfallsicher sein wie ein Telefonnetz. Es muss durch Vervielfachung der CSCF- und HSS-Instanzen skalieren können von 100.000 bis zu mehreren 10.000.000 Teilnehmern. Die NGN möglichst aller Netzbetreiber müssen kontrolliert und sicher zusammenarbeiten können, um Dienste ähnlich wie heute die Mobiltelefonie weltweit verfügbar zu machen.

Wie fast alle Konkurrenten auch, entwickelt Nokia Siemens Networks zwei separate Hauptprodukte, die als mehrere CSCF- bzw. HSS-Instanzen ein IMS-Kernnetz bilden können. Durch die ausschließliche Ausrichtung auf Paket-vermittelte Kommunikation mittels IP (Internet Protocol) war es möglich, Standard-Hardware (Server, Blades) zu verwenden und die IMS-Funktionalität praktisch ausschließlich in Software zu realisieren. Die Basis der Software ist eine Plattform, die auch in anderen Produkten zum Einsatz kommt. Diese Plattform stellt grundlegende Funktionen zur Absicherung der Hochverfügbarkeit, für die Verteilbarkeit/Skalierbarkeit. sowie für das Netzelement-Management bereit. Mit einem Linux-Betriebssystem als Basis verwendet sie Standard-Softwarekomponenten wo immer es möglich und wirtschaftlich sinnvoll ist.

Die IMS Software ohne Plattform hat inzwischen einen Umfang von über 2 Mio. Netto Lines of Code (NLOC). Sie ist überwiegend in C++ sowie teilweise in Java programmiert.

3 Warum agile Entwicklung?

Nicht zuletzt aufgrund der wachsenden Konkurrenzsituation unter den IMS-Anbietern war es nötig, bei Nokia Siemens Networks die Time-to-Market zu reduzieren, konkret: schneller als bisher auf neue bzw. geänderte Kunden-Anforderungen reagieren zu können.

Hinzu kamen einige Spezifika von IMS: IMS ist ein sehr flexibel einsetzbares und ein recht neues Produkt, für das die gültigen Standards noch ständig erweitert werden. Das bedeutet, dass – wesentlich stärker als bei traditionellen Telekommunikations-Systemen – die konkrete Marktausrichtung viel Fingerspitzengefühl im Produktmanagement sowie eine schnelle Reaktion auf Kundenanforderungen und Markttrends erfordert. Dadurch waren wir als IMS-Entwicklung aufgefordert, unsere Konzepte und Prozesse so zu ändern, dass neue Features in guter Qualität sehr schnell geliefert werden konnten, um sie in Trials, auf einem "Test Fest" oder auch bei kommerziellen Interessenten demonstrieren zu können.

Zusätzlich bestand die Forderung nach Auslieferung von drei Produktfreigaben pro Jahr, d.h. eine Hauptversion und zwei Unterversionen.

4 Spezifische Herausforderungen des IMS-Projekts

Die IMS-Entwicklung ist wegen ihrer Größe – zeitweise 180 Mitarbeiter – und zur Reduzierung der Kosten geographisch über mehrere Entwicklungsstandorte in Deutschland, Österreich, Tschechische Republik, Indien und China verteilt. Eine enge Interaktion über alle Standorte hinweg ist hier mangels unmittelbarer und direkter Kommunikation praktisch nicht umsetzbar. Wir haben daher die bestehende Zuordnung von Funktionsbereichen bzw. Subsystemen zu Standorten beibehalten. Innerhalb der Standorte konnte allerdings eine flexiblere Umverteilung bzgl. Personen und Know-How zwischen den Teams ermöglicht und erreicht werden. Die hinreichend frühe und präzise Spezifikation von Anforderungen und der Systemarchitektur schien uns weiterhin als das geeignetste Mittel für eine funktionierende Aufteilung der Aufgaben über die Standorte.

5 Agile Analyse und Design für die IMS Entwicklung

Als wir mit unseren Überlegungen zur agilen Transformation begannen, hatten wir (bis IMS Version 6) im Wesentlichen ein Wasserfall-Modell verwendet, jedoch mit einer wichtigen Besonderheit, die uns den Übergang zu agilen Methoden sehr vereinfachte: die sog. *Integrationsstufen*.

Das bedeutete, dass es auch in frühen IMS-Versionen nicht ein einziges Zusammenführen aller Modul-getesteter Komponenten gab, sondern es gab mehrere Integrations-Zeitpunkte, zu denen möglichst früh Teile der definierten Funktionen zusammengefasst und getestet wurden. Diese Vorgehensweise vereinfachte die Einführung von Agilen Techniken, sowohl die Definition der Analyse-Phase als auch den Weg zur Continuous Integration (CI).

5.1 Dokumentations-Konzept im früheren Wasserfall-Prozess

Im Wasserfall-Prozess verwendeten wir folgende Dokumente:

- **Feature Requirements Specification, FRS:**
  Für jedes Produktfeature entsteht ein solches Dokument bestehend aus zwei Hauptteilen:
  1) Die Requirements (das sog. *chapter 7* bzw. *ch.7*): Black-Box-Darstellung der Details dieses Features. (Nach dem V-Modell ist dieser Teil auch Basis für die System-Tests.)
  2) Die architekturelle Lösung (das sog. *chapter 9* bzw. *ch.9*): Es handelt sich hier um eine relativ grobe Beschreibung der Grundidee und der Schnittstellen. Diese Beschreibung sollte gut genug sein, um auf dieser Basis Aufwandsabschätzungen durchführen zu können.
  Ein FRS hat – je nach Feature – einen Umfang von ca. 10 bis max. etwa 100 Seiten.

- **Functional Specification Level 0 (F-SpecL0):**
  Beschreibung der Gesamt-Systemarchitektur einer IMS-Version, als wichtige Basis für die Schulung der Vertriebe sowie für die Kundendokumentation.

- **Functional Specification (F-SpecL2) und Interface-Specification:**
  Eine F-SpecL2 beschreibt die funktionale Architektur eines von mehreren IMS-Subsystemen. Die Detaillierung muss hinreichend für die Ableitung einer ausführlichen Schnittstellen-Definition sowie für Implementierung des beschriebenen Subsystems sein.
  Grundsätzlich erwies sich die starre Einbindung der Erstellung dieser Dokumente in den Prozess als allzu unflexibel bzgl. kurzfristiger, schneller Änderungswünsche, was sich dann besonders beim Übergang auf die agile Vorgehensweise zeigte. Wir machten aber auch mehrfach die Erfahrung, dass eine zu leichtgewichtige Behandlung der funktionalen Architektur für solche Änderungen schnell zu unzureichend durchdachten Lösungen führen kann

- **Design-Specification (D-Spec):**
  Diese Dokumente beschreiben die wichtigen Prinzipien der Implementierung.
  Auch hier bestand ein immanentes Problem: Die D-Specs entstehen bzgl. Zeit und Dokumentenverwaltung separat vom Code. Die dadurch praktisch unvermeidbare Divergenz von Code vs. Dokumentation musste explizit adressiert werden durch eine gezielte Aktualisierung dieser Dokumente kurz vor dem System-Test.

5.2 Dokumentations-Konzept in der Analyse-Phase (agiler Prozess)

Zu Beginn der Diskussionen um unsere agilen Konzepte war die Grundidee, sämtliche der oben genannten Dokumente abzuschaffen. Der wichtigste Grund war, dass alle diese Dokumente im Prinzip die Flexibilität einschränkten, d.h. den Aufwand bei Änderungen erhöhten, was ganz klar einem zentralen Vorteil von agilen Prozessen widerspricht.

Bei detaillierter Betrachtung und Diskussion der Prozess-Semantik stellten wir jedoch fest, dass ein gewisser Mangel an Flexibilität sogar nützlich ist und gewollt sein kann: Häufige Änderungen der Anforderungen führen zu nicht-dokumentierter Zusatzarbeit, die der eigentlichen Entwicklung dann fehlt. Damit kommt man sehr schnell zu der (banalen) Erkenntnis, dass damit einer der wichtigsten Vorteile von Wasserfall-Prozessen – die klare Einigung zu der Bestellung eines Features – aufgegeben worden wäre. Letztlich kamen wir überein, dass (a) Änderungen der Anforderungen möglich sein sollten, dass aber (b) die erhöhten Aufwände erkennbar sein sollten, und dass wir den damit verbundenen Zusatzaufwand (Overhead) minimieren wollten.

Als Resultat dieser Überlegungen behielten wir die Feature Requirements Specifications (FRS) im Wesentlichen unverändert bei. Wir konnten sogar das Qualitäts-Kriterium für die FRS klarer fassen: Eine FRS ist dann fertig – d.h. sie erfüllt das agile "done"-Kriterium –, genau dann wenn sie als Basis für Aufwandsabschätzungen dienen kann, d.h. "done = Aufwände liegen vor". Damit wurden die FRS als belastbare Referenz für die nächsten Phasen definiert, die idealerweise mehrere Sprints überdecken sollte. Damit können auch Änderungen der Anforderungen an genau dieser Stelle vereinbart, dokumentiert und der resultierende Aufwand abgeschätzt werden.

Eine agile iterative Weiterentwicklung komplexer Features und der betroffenen FRS ist möglich. Eine solche FRS-Iteration kann die Vorversion sowohl erweitern, als auch frühere Anforderungen oder Architekturkonzepte verändern, Der resultierende Zusatzaufwand inkl. des Änderungsaufwandes in allen Folgephasen wird umgehend sichtbar, und kann im Zweifelsfall ein Überdenken der Wirtschaftlichkeit teurer Änderungen in Relation mit den übrigen offenen Features anstoßen.

Die F-SpecL0 verwendet weiterhin die FRS als Basis. Sie beschreibt letztendlich das Ergebnis aller Sprints einer verkaufbaren Produktversion.

Die F-SpecL2 waren weit problematischer: Weil sie einerseits – als Resultat gründlicher Analyse – wichtig waren für eine hohe Qualität der Implementierung, sie aber andererseits damit auch sehr änderungsunfreundlich waren, war es lange unklar, ob und in welcher Form sie beibehalten werden sollten.

Schließlich nahmen wir die Verwendung und Relevanz dieser Dokumente für spätere Phasen als Basis für unsere Überlegungen. Damit schälten sich zwei Aspekte heraus: (1) Fixierung der Interfaces, und (2) detaillierte Analyse der Subsysteme.

Fixierung der Interfaces: Hier stellten wir fest, dass die wichtigste Vereinbarung zwischen den beteiligten Gruppen der Entwicklung die Header-Files (die Klassen-Definitionen in C++) sind. Hinzu kommt trivialerweise, dass die syntaktische Korrektheit sowie die Konsistenz dieser Dokumente mit einem Compiler leicht zu überprüfen ist. Die fehlerfreie Kompilierbarkeit dieser Header-Files wurde also als wichtiges Meilenstein-Kriterium für "Ende der Analyse" übernommen. Die Interfaces sind somit das wichtigste Konzept, das über mehrere Sprints möglichst stabil gehalten wird. Datenbank-Schemata sind im IMS ebenfalls durch Interfaces repräsentiert, so dass deren Stabilität ebenfalls gesichert ist.

Detailierte Analyse der Subsysteme: Selbstverständlich muss für eine Implementierung in hoher Qualität die Analyse für die betroffenen Module abgeschlossen sein. Andererseits umfasst eine "klassische" F-SpecL2 die *gesamten* Änderungen bzw. Neu-Implementierungen, die für ein Feature erforderlich sind, also mithin mehrere Sprints. Hier konnte es also hinderlich – bzw. zu wenig agil – sein, zu einem Meilenstein die gesamten Dokumente vollständig fertig haben zu wollen. Dass die F-SpecL2 jedoch erforderlich waren für die hohe Qualität der Implementierung, das war für uns unstrittig. Diese Überlegungen resultierten in der Entscheidung, dass die F-SpecL2 weiterhin erstellt werden, dass sie aber zum Meilenstein *noch nicht vollständig* vorliegen müssen; sie sollten jedoch mindestens den Inhalt der nächsten beiden Sprints enthalten. Das bedeutet aber umgekehrt, dass diese Dokumente als "lebende Dokumente" angesehen werden, d.h. dass sie sich in der Entwicklung-Phase verändern können (und sollen).

5.3 Dokumentations-Konzept in der Design-Phase (agiler Prozess)

Die massivste Änderung nahmen wir bei den **Design-Specifications (D-Spec)** vor: Da deren wichtigstes Ziel ist, die Implementierung zu dokumentieren, schlugen wir vor, die Inline-Dokumentation als Basis zu verwenden; das sollte auch dazu beitragen, das oben genannte Risiko der "Divergenz Code vs. Dokumentation" zu reduzieren. Zu diesem Zweck führten wir das bekannte Tool Doxygen ein: Es erlaubt, aus Kommentaren im Code eine Dokumentation in HTML zu erstellen. Darüber hinaus dokumentiert es für den C++-Code auch übergreifende Strukturen, z.B. die Vererbungshierarchie.

Damit kamen wir aber zu einer weiteren Schwierigkeit: Da sich bei agiler Entwicklung der Code und die dazugehörige Inline-Dokumentation ständig verändern, kann es schwierig sein, einen bestimmten Stand des Codes und dessen Dokumentation zu erhalten. Für diesen Zweck führten wir einen bestimmten Stichtag (Beginn des System-Tests) ein, zu dem die generierten HTML-Files eingefroren werden, und in einem "statischen" Zweig zur Verfügung stehen. Dieser Stand dient im Übrigen auch als Referenz für Reviews und für die Aktualisierung der F-SpecL2.

Änderungen an Code und Inline-Dokumentation können mit diesem Konzept – auf dem dynamischen Zweig – weiterhin vorgenommen werden, und jeder Entwickler kann sich jederzeit den aktuellsten Stand von Doxygen generieren lassen.

Erst später fiel uns auf, dass wir bei dieser Vorgehensweise das früher in den D-Specs verlangte Requirements-Tracing zunächst „vernachlässigt“ hatten. Eine explizite Verpflichtung zur Integration des Tracings in Doxgen-Kommentare sowie eine Anpassung der Werkzeugkette empfanden die Entwickler als ärgerlich—vielleicht auch, weil sich der Wert des Tracings bis ins Design in der bisherigen Praxis nicht so deutlich gezeigt hat.

6 Der agile Planungs-Prozess

Ein wichtiges Erfolgskriterium für eine agile IMS-Entwicklung war eine funktionierende Zusammenarbeit zwischen Produktmanagement, Produktarchitekten und der Software-entwicklung. Das hierfür entwickelte Vorgehen zeigt Abbildung 1, in der die Besprechungen als dunkle Rechtecke eingezeichnet sind.

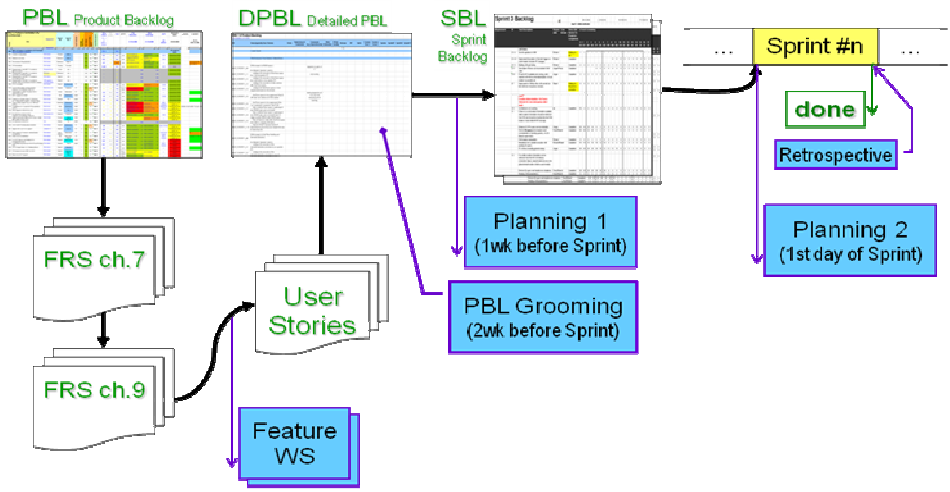

Abbildung 1: Dokumente im agilen Planungsprozess

In der **PBL** (Product Backlog, siehe Abbildung 2) sind die Features mit Prioritäten aufgelistet. Für jedes dieser Features existieren eine FRS (Feature Requirements Specification) und damit auch eine Aufwandsabschätzung. Auf dieser Basis kann nun eine verkaufbare IMS-Produktversion paketiert werden, d.h. deren wesentliche Inhalte beschlossen und an den Vertrieb kommuniziert werden. Für Änderungen wird ein bestimmter Anteil des verfügbaren Gesamt-Aufwands der Entwicklung (z.B. 30%) vorgehalten.

Die Entwicklung kann ebenfalls neue Features vorschlagen -- sowohl Kundenfeatures, z.B. eine verbesserte Bedienbarkeit, als auch strukturelle Veränderungen wie Refactorings oder die Entfernung überflüssig gewordener Funktionen. Solche Features werden im Prozess vollständig gleichberechtigt neben denjenigen des Produktmanagements behandelt.

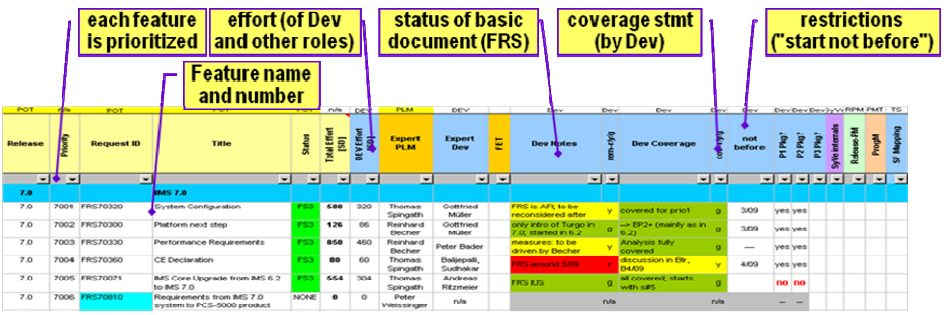

Abbildung 2: Beispiel einer PBL (Ausschnitt)

Aus den FRS werden in Feature Workshops nun **User Stories** erstellt und in der **DPBL (Detailed PBL)** gesammelt (siehe Abbildung 3).

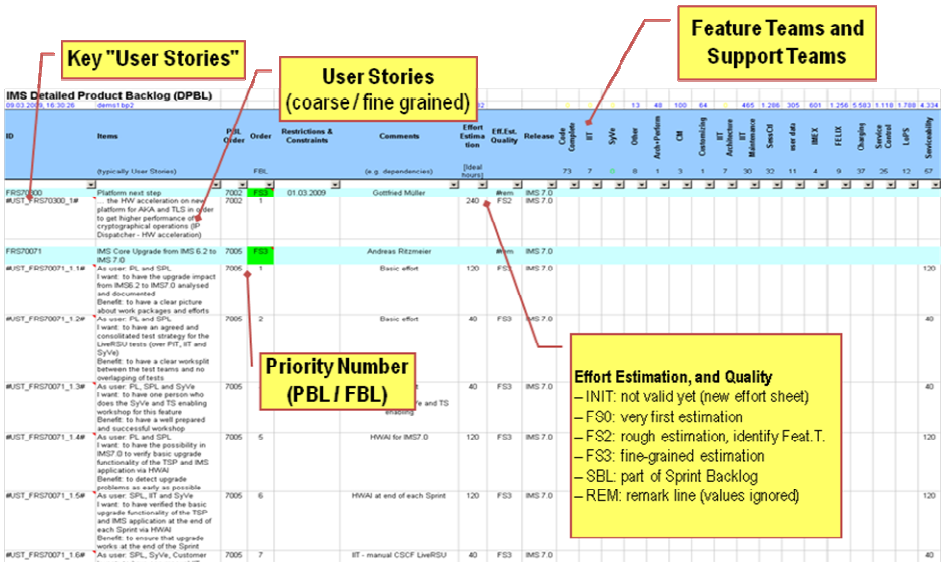

Abbildung 3: Beispiel einer DPBL

Eine DPBL enthält sämtliche User Stories, je nach Ausarbeitung detailliert (*fine grained*, für die nächsten beiden Sprints) oder grob (*coarse grained*, für spätere Sprints). Hinzu kommen die Aufwandsabschätzungen, in Summe und für jedes beteiligte Team, d.h. sowohl für die implementierenden Teams (Feature Teams) als auch für übergreifende Teams, z.B. für Bring-Up oder Test (Support Teams). Die DPBL wird automatisch aus allen **Feature Backlogs (FBL)**, die von den Feature Teams erstellt werden, zusammengesetzt. Die daraus resultierende Tabelle sortiert die Features gemäß ihrer Priorität im PBL, summiert die Einzelaufwände zu einem Feature-Gesamtaufwand, und zeigt alle an einem Feature beteiligten Teams (siehe Abbildung 4).

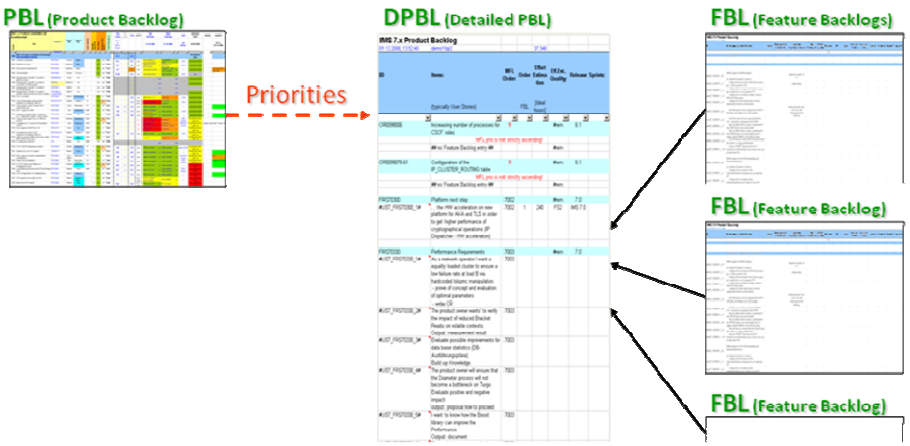

Abbildung 4: Aufbau einer DPBL aus FBLs

Im **PBL Grooming Workshop** werden diese User Stories den folgenden vierwöchigen Sprints zugeordnet. Dabei gilt, dass die Zuordnung zu den nächsten beiden Sprints weitgehend verbindlich sein soll, für die nachfolgenden Sprints reicht eine grobe Zuordnung; deren wichtigster Zweck ist, *Restriktionen* in der Implementierungs-Reihenfolge zu dokumentieren.

Die Planung für einen konkreten Sprint findet eine Woche vor diesem statt und mündet in die **SBL (Sprint Backlog)**. Am ersten Tag jedes Sprints findet die konkrete Zuordnung zu Personen statt, so wie das bei agilen Prozessen allgemein bekannt ist.

Änderungen durchlaufen diese Prozess-Kette auf folgende Weise:

- In einem ersten Schritt wird im **Product Owner Team (POT)** entschieden, ob und welche neue Features implementiert werden sollen. Das POT trifft sich 14-täglich, und in ihm sitzen Vertreter von Produktmanagement, Projektleitung, Vertrieb, Produktarchitektur, Softwareentwicklung und Systemtest.

- Wird ein neues Feature akzeptiert, dann wird eine **FRS** (mit Aufwandsabschätzung) beauftragt.

- Sobald die FRS, die Aufwände und eine Stellungnahme über den Marktwert des betrachteten Features vorliegen, kann – ebenfalls im POT – die Implementierung beauftragt werden.

- Wenn ein Feature als sehr dringend angesehen wird, und damit ein anderes verdrängt oder tiefer priorisiert wird, dann wirkt sich das unmittelbar auf die Prioritäten in der DPBL aus. Diese Änderung wird also im nächsten PBL Grooming Workshop behandelt und in den nächsten Sprint eingeplant.

- Für sehr wichtige Features kann – *im Ausnahmefall!* – die FRS-Erstellung bereits angestoßen werden, so dass im POT nur über die Implementierung diskutiert wird. Damit können sehr wichtige – *und wenig komplexe!* – Features in ca. 6 Wochen in Trial-fähiger Qualität vorliegen.

7 Erfahrungen

Der agile Softwareentwicklungsprozess wird seit 2009 in der IMS-Entwicklung auf voller Breite und mit Erfolg eingesetzt. Die Planbarkeit und die Flexibilität bzgl. neuer und geänderter Anforderungen haben sich wesentlich verbessert. Wir sehen hier folgende Hauptfaktoren für den Erfolg:

A) Die formalisierten Absprachen zwischen Produktmanagement, Systemarchitekten und Softwareentwicklung mit den resultierenden Product Backlogs haben zu einer deutlich schnelleren und letztendlich sogar zuverlässigeren Versionsplanung geführt.

B) Die Continuous Integration (CI) vermeidet wiederholte Systemdestabilisierungen und reduziert damit Zeit und Aufwand.

C) Die optimierte Dokumentenstruktur unterstützt die rechtzeitige und „passgenaue“ Kommunikation von Teams über die Standorte hinweg, und ist damit ein wichtiger Baustein für die Qualität der Software.

Die verschiedenen Entwicklungsstandorte waren von Anfang an bei der Definition der neuen Prozesse sowie bei den Planungen für deren Einführung beteiligt. Die Akzeptanz bei praktisch allen Entwicklern ist sehr hoch – einerseits wegen eines breiten Interesses an den „angesagten“ agilen Methoden, andererseits wegen der direkteren Beteiligung der Teams und der einzelnen Entwickler an den Planungsprozessen.

Es gab allerdings auch einige eher unerwartete Erkenntnisse:

- Die Projektstruktur ist komplexer geworden, womit dann auch der Aufwand für Steuerung und Kontrolle gestiegen ist, weil mehr Kennwerte erhoben und ausgewertet werden müssen.

- Die höhere Flexibilität beim agilem Vorgehen hat zu einer Erhöhung des Aufwands pro Feature geführt (ähnlich wie bei Realzeit-Systemen, die in der Regel eine niedrigere Performance aufweisen)

- Wichtige Features können damit sehr schnell implementiert werden; kurzfristige Änderungen sollten jedoch die Ausnahme bleiben, weil der damit verbundene Zusatzaufwand schon allein für eine frühe Analyse andere, möglicherweise wichtigere Features gefährdet (selbst dann, wenn "eigentlich" genügend Aufwand übrig wäre). Die agile Vorgehensweise ermöglicht allerdings eine sachliche Diskussion und Entscheidung über die Wirtschaftlichkeit und Priorität von einzelnen Änderungen.

Die wichtigste Erfahrung war sicher, dass sich ein Nutzen aus der Einführung agiler Softwareentwicklung auch dann gewinnen lässt, wenn man nicht alle agilen Prinzipien umsetzen kann. Schon alleine das offiziell geforderte Überdenken der bestehenden Prozesse führte in unserem Fall zu einer stärkeren Beteiligung der Entwickler (und damit zu einer höheren Identifikation mit dessen Regeln), zu einer sinnvolleren Dokumentenstruktur und schließlich auch zu besserer Qualität des Codes.

8 Ratschläge

Aus unserer Erfahrung lässt sich sagen, dass sich die Einführung von agilen Methoden in die IMS-Prozesse generell gelohnt hat. Als Erfolgsfaktoren lassen sich u.a. nennen:

- *Kein dogmatisches Vorgehen bei Einführung von agilen Prozessen!* Wenn irgendein Aspekt nicht klar ist (oder gar hinderlich erscheint), dann sollte dieser Prozess-Aspekt zunächst vermieden werden. Appelle wie "Wenn wir es nicht so machen, dann sind wir nicht richtig agil!" sind nicht nur kontraproduktiv, sondern rühren in manchen Fällen von einem falschen Verständnis des Prozesses her.

- *Einbinden der Entwickler!* Wenn diejenigen, die mit den neuen Methoden arbeiten sollen, nicht an – zumindest grundsätzlichen – Überlegungen beteiligt werden, dann führt das mindestens zu häufigen Missverständnissen bei der Umsetzung, bis hin zum weitgehenden Ignorieren des Neuen. Wir hatten bei IMS auch des große Glück, viele Entwickler zu haben, die sich für diese Themen interessierten und sehr gute Vorschläge beisteuerten.

- *Die Hürden – wenn möglich – niedrig halten!* Manchmal sind größere Änderungen unvermeidbar. Aber oft kann man auch schrittweise vorgehen, und man erhält auf diese Weise viel schnelleres Feedback, wenn ein falscher Weg beschritten wurde. (Die Analogie zur agilen Entwicklung liegt auf der Hand.)

- *Ausnahmen* (z.B. beschleunigte Sub-Prozesse für sehr dringende Features) *sollten Ausnahmen bleiben!* Wenn ein Prozess viele Ausnahmen braucht, dann ist er in der Regel nicht sehr gut (und auch nicht mehr nützlich).

Literaturverzeichnis

[Boe03] Boehm, B.; Turner, R.: Balancing Agility and Discipline. Addison-Wesley Longman, Amsterdam, 2003.

[LV10] Larman, C.; Vodde, B.: Practices for Scaling Lean and Agile Development. Addison-Wesley Longman, Amsterdam, 2010.

[Sch04] Schwaber, K.: Agile Project Management with Scrum. Microsoft Press. 2004.

[WRL05] Wolf, H.; Roock, S.; Lippert, M.: eXtreme Programming: Eine Einführung mit Empfehlungen aus der Praxis. Dpunkt Verlag, Heidelberg, 2005.