Erfolgreiche Tradition trifft dynamische Moderne

Einführung von anforderungs- und modellgetriebener Entwicklung

Kai Gloth, Sartorius Lab Instruments

Für jeden Teil der Produkt- bzw. Instrumentenentwicklung und der damit verbundenen Prozesse gibt es unendlich viele Bücher und Ratgeber. Suche ich auf Amazon nach dem Begriff *Requirements Engineering* werde ich mit weit über 2000 Ergebnissen konfrontiert. In den Standardwerken, Klassikern und Neuerscheinungen ist jeder Aspekt berücksichtigt. Für jedes noch so kleine Detail gibt es Ausarbeitungen von mehreren Seiten, die alle - vorausgesetzt sie werden im Detail gelesen - den Leser auf jede Situation vorbereiten sollen. Genau dies habe ich gemacht. Als ich im Jahr 2014 die Aufgabe bekommen habe die Neuentwicklung eines Geräts zu starten, habe ich nahezu jeden Teilaspekt der Entwicklung in Frage gestellt und mit Hilfe von Büchern versucht zu erarbeiten, wie diese Teilschritte und die gesamte Entwicklung verbessert werden können. Mein Plan war es, insbesondere Requirements Engineering und so weit wie möglich modellgetriebene Entwicklung in allen Bereichen zu etablieren.

Dieser Artikel beschäftigt sich mit den Dingen, die ich nicht in Büchern gelernt habe. Er handelt von den Problemen, mit denen ich in der täglichen Arbeit konfrontiert wurde. Manchmal konnte ich die Probleme mit Erfolg lösen, manchmal aber auch überhaupt nicht.

Das Team, mit dem ich gearbeitet habe, hat die Vorgänger der Geräte erfolgreich entwickelt, bestand nahezu nur aus Senior-Entwicklern und war zum Teil schon seit mehreren Jahrzehnten im Unternehmen. Aus Sicht des Entwicklungsteams gab es also keinen Anlass, die Entwicklung grundlegend zu ändern. Der Produktentwicklungsprozess in der Instrumentenentwicklung folgt dem V-Modell. Allerdings sollten Elemente der agilen und iterativen Entwicklungsprozesse wiederverwendet werden. Das gesamte “V” wird also nicht nur einmal wasserfallartig durchlaufen, sondern folgt einer Iteration über einen bestimmten Zeitraum. Der Übersichtlichkeit halber orientiert sich dieser Artikel ebenfalls an den Ebenen des V-Modells und beleuchtet die Teilaspekte.

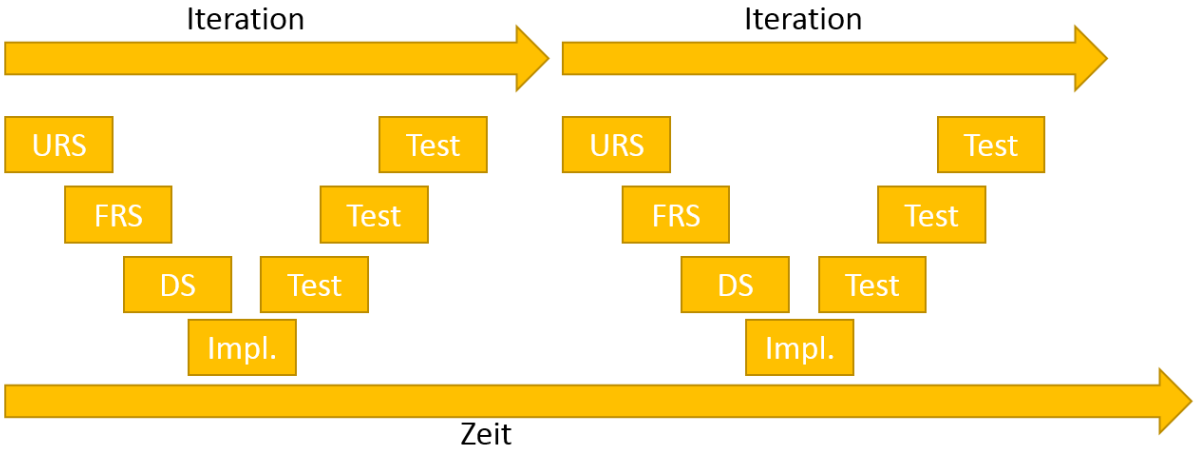

Abbildung 1: V-Modell iterativ

User Requirements Specification - URS
Die URS stellt die Anforderungen an das Produkt bzw. Instrument aus der Sicht der Benutzer dar. Jeglicher Stakeholder, also die Person bzw. Personengruppe, die ein Interesse an dem Produkt hat, sollte erfasst, berücksichtigt und priorisiert werden. Requirements Engineering ist nicht kompliziert oder aufwendig. Meiner Erfahrung nach besteht die Disziplin des Requirements Engineering aus zwei Teilbereichen:

Der erste Bereich beschäftigt sich mit der Erstellung des Metamodells oder auch Informationsmodells. Das bedeutet welche Information hat welche Wertigkeit, Eigenschaften und wie wird damit weiter verfahren. Ein Ideenpool zum Beispiel. Jede Idee hat nur eine Überschrift und wird nach Schulnoten bewertet. Ideen stellen aber keine Anforderungen an das Produkt dar, sondern unterstützen nur den Prozess der Anforderungsfindung.

Der zweite Bereich umfasst das eigentliche Verfassen, Prüfen und Gegenlesen von Anforderungen. Meiner Erfahrung nach sollte ein Informationsmodell möglichst einfach gehalten und für jeden verständlich sein, also insgesamt wenig Artefakte und Regeln beinhalten. Dies hat gleich mehrere Vorteile. Zum einen verringert es die Schwelle für das einzelne Teammitglied Anforderungen zu formulieren, zum anderen bleibt es so flexibel, dass lokale Anpassungen durchgeführt werden können. Anforderungen auf Ebene der Benutzer werden häufig dazu verwendet interne Konflikte auszutragen insbesondere, zwischen den Abteilungen Marketing, des Product Owners und dem R&D Bereich aber auch, um jegliche Details in den Anforderungen zu erfassen bzw. zu fordern. Wird Requirements Engineering als Disziplin eingeführt ist es wichtig, dass Verständnis für das Detaillevel bei allen Beteiligten zu schärfen und kontinuierlich zu überprüfen. Der Level an Detail ist deshalb so schwierig, weil es häufig kein klares Ja oder Nein gibt. Ein Detail, wie z.B. die Verwendung der Technologie oder eines bestimmten Edelstahls für ein Gehäuse, kann auf den ersten Blick aus Benutzersicht völlig irrelevant sein, auf den zweiten Blick kann es sich aber auch um eine allgemeine Voraussetzung oder konkrete Anforderung von bestimmten Kunden handeln.

Ich empfehle nicht dogmatisch vorzugehen! Was ein Team eigentlich erreichen möchte ist ein gemeinsames Verständnis der Anforderungen, ohne unnötigen Aufwand auf Seiten der Anforderungsgeber als auch der Anforderungsumsetzung. Anforderungen sollten lösungsneutral formuliert sein. Gibt es aber einen bewussten Grund eine Lösung konkret zu formulieren, dann ist das durchaus erlaubt. Nehmen sie das Team in die Pflicht Anforderungen aktiv mitzugestalten. Das Verständnis von Benutzeranforderungen ist bereits der erste Teil der Entwicklungstätigkeit. Weiterhin empfehle ich ein gutes Tooling, um Anforderungen möglichst einfach zu erfassen und auch Änderungen über die Zeit nachverfolgen zu können. Theoretisch möglich wäre die Verwendung von Officetools, von denen würde ich allerdings abraten. Gute Erfahrungen habe ich mit dem Team Foundation Server und Polarion gemacht.

Functional Requirement Specification – FRS Die User Requirement Specification (URS) beschreibt die gewünschten Anforderungen seitens der Stakeholder. Die Functional Requirement Specification (FRS) beinhaltet die Antwort oder auch den ersten Lösungsansatz seitens des Entwicklungsteams. Im Amtsdeutsch würde man zur URS bzw. FRS auch Lasten- und Pflichtenheft sagen. Wichtig hierbei zu verstehen ist, dass sich die Verantwortung umkehrt. Verantwortlich für die URS ist

der Stakeholder oder auch Product Owner und es muss sichergestellt sein, dass das Entwicklungsteam die Anforderungen versteht. Die Verantwortung für die FRS liegt bei dem Entwicklungsteam. In diesem Fall muss sichergestellt sein, dass der Stakeholder, Benutzer auch auch Product Owner der vorgeschlagenen Lösung zustimmt und diese auch versteht.

Meiner Erfahrung nach eignet sich dieses Level hervorragend um mit einer Modellierungssprache wie UML zu beginnen. Insbesondere Use Case und Activity Diagramme bieten eine gute Plattform um das vorgeschlagene System funktional zu strukturieren und mit wenigen Worten und Elementen eine Einigung über umzusetzende Funktionalitäten herbeizuführen. Gefährlich ist eine Überdetaillierung der Diagramme. Ziel ist es eine gewünschte Funktionalität aufzuzeigen, ohne auf technische Details hinzuwiesen.

Wichtig ist außerdem, dass ein funktionaler Schnitt des Systems nicht gleichzusetzen ist mit dem technischen Schnitt ist. Ein Use Case Diagramm kann zum Beispiel alle Funktionen des User Managements beschreiben, inkl. dem Anlegen neuer Benutzer, während die technische Umsetzung eine Teilung der Funktion auf mehrere Teilsysteme, wie z.B. User Management, Database Layer und Authentication Service vorsieht.

Im Bereich embedded Systeme empfehle ich ein Design und eine Erarbeitung von Funktionen unabhängig der Disziplin. Um bei dem Beispiel des User Management zu bleiben, sehe ich auch die Funktion der Authentifizierung durch Fingerabdruck als Bestandteil der Gruppe.

Wird ein ALM Tool wie Polarion oder Team Foundation Server verwendet, zeigt sich hier zum ersten Mal die Stärke eines solchen Tools. Requirements aus der URS können mit Elementen der FRS verbunden werden, wenn sich diese aus den Requirements ergeben bzw. abgeleitet wurden. Werden Requirements verändert kann ich durch die baumartige Struktur erkennen, welche Elemente in der FRS beeinflusst werden.

Design Specification - DS

Die Design Specification beinhaltet die konkrete Dokumentation über die technische Umsetzung. Sowohl die URS als auch FRS dienen hauptsächlich der Abstimmung über das „Was“. Die DS beinhaltet das konkrete „Wie“.

Meiner Erfahrung nach ist dieses Dokument sehr gut geeignet um Abstimmungen, insbesondere bei großen Teams, durchzuführen. Ich würde das Dokument daher als wachsendes und sich ständig veränderndes Dokument ansehen, bis eine Änderungskontrolle innerhalb des Projekts notwendig wird. Ich empfehle, den Großteil der Kommunikation auf UML bzw. SysML zu beschränken und möglichst wenig geschriebenes Wort zu verwenden. In der Praxis bewährt haben sich zum Beispiel Component, Use Case, Class und Object Diagrams, aber auch Block Definition, Internal block oder Package Diagrams. Dieses Zusammenspiel von verschiedenen Typen ermöglicht eine gemeinsame Betrachtung aller Disziplinen in zusammenhängenden Diagrammen. Mit Hilfe von Component Diagrams ist es unter anderem möglich zu zeigen, welcher Softwareteil welche Aktorik bzw. Sensorik ansteuert oder auch welcher Softwarebaustein auf welchem Prozessor ausgeführt

wird. Insbesondere eine gemeinsame Dokumentation, unabhängig der technischen Disziplin, ist ein nicht zu unterschätzender Vorteil. Es weckt auf allen Seiten das Verständnis für Änderungen, Zusammenhänge und Komplexität, erlaubt eine bessere Risikoabschätzung und gibt den Testern im Team ein umfangreicheres Bild.

Ich empfehle, den Fokus bei der Verwendung von UML und SysML zunächst nur auf Aspekt der Dokumentation zu legen. Häufig wird direkt nach dem Schlüsselwort UML sofort der Begriff Codegenerierung verwendet. Meiner Erfahrung nach bedeutet dies einen unwesentlichen, und in vielen Fällen auch ungerechtfertigten Mehraufwand, den ein Entwickler bei der Erstellung von den Diagrammen investieren muss.

Insgesamt empfehle ich kontinuierlich den Detailgrad der Dokumentation und insbesondere der Diagramme zu hinterfragen. Meiner Erfahrung nach hat Dokumentation das Ziel, die Kommunikation zwischen Entwicklern zu vereinfachen, generelle Konzepte zu verdeutlichen, als auch den Einstieg von Fremd- oder Neuentwicklern zu vereinfachen. Eine Detaillierung darüber hinaus halte ich nicht gerechtfertigt und, vorausgesetzt der Einsatzbereich bzw. Markt schreibt es nicht vor, auch für unnötig.

Ich beobachte häufig Situationen in denen der erste Entwurf der Dokumentation hervorragend verfasst und geschrieben ist. Insbesondere aber über die Zeit und wenn der Projektdruck steigt, sich immer weiter von der tatsächlichen Implementierung entfernt. Als hervorragend geeignet hat sich eine Code- und Architekturüberprüfung wie z.B. Axivion herausgestellt. In regelmäßigen Abständen überprüft ein solches Tool den Code und gleicht ihn mit der Dokumentation ab. Als Input verwendet Axivion das UML. Dies erlaubt den Team eine Aussage zur Qualität, der Implementierung als auch Dokumentation zu treffen und schafft Vertrauen, wenn Software von dritten auditiert wird.

Zur Erstellung von UML Diagrammen empfehle ich Sparx Enterprise Architect. Auf dem Markt erhältlich sind außerdem Konnektoren zwischen ALM Tools, wie Polarion oder dem Team Foundation Server.

Testing und Risikomanagement

Der Bereich des Testing und des Risikomanagement findet in diesem Artikel keine besondere Erwähnung. Die von mir gemachte Erfahrung unterscheidet sich nicht zu anderen Bereichen des Artikels. Insbesondere die Dokumentation mit Hilfe von UML oder SysML war seitens des Testteams ein hilfreicher Input und hat die Arbeit stark vereinfacht. Weiterhin empfehle ich das Testteam als integraler Bestandteil des Teams anzusehen. Das Feedback seitens eines Testers bzw. Testdesigners ist auf allen Ebenen, begonnen bei der URS bis hin zur DS, überaus wichtig und sollte kontinuierlich berücksichtigt werden.

Das Risikomanagement wird ebenfalls mit Hilfe einer gut aufgestellten Dokumentation vereinfacht. Änderungen und Auswirkungen können besser nachvollzogen werden und wichtigen Input, sowohl für die Bewertung des Risikos als auch für die Testumfänge, bilden.

Autor

Kai Gloth arbeitet momentan als R&D Systems Engineer und betreut internationale interdisziplinäre Projekte und Teams. Hierbei legt er insbesondere Wert auf eine transparente Entwicklung und eine gemeinsame technische Sprache der unterschiedlichen technischen Disziplinen und Stakeholder.