Stefan Mieth

(Stefan.Mieth@aitgmbh.de)
ist Software Process Consultant der AIT GmbH & Co. KG. Er hilft Teams bei der Verbesserung der Softwareentwicklung und berät bei der Einführung und Anpassung des Visual Studio Team Foundation Server. Er hat sich auf die Bereiche Projektmanagement, Entwicklungsprozesse und Anforderungsmanagement spezialisiert. Seine Erfahrungen vermittelt er zudem als Autor des TFS-Blogs, in Magazinen und in Vorträgen. Das AIT-Team besteht aus Experten rund um Softwareentwicklung, Team Foundation Server und Application Lifecycle Management.

„Früher war alles besser" – eine Reise zu längst vergessenen Tagen

Vor kurzem fielen mir meine alten Unterlagen aus Studienzeiten in die Hände. Beim genaueren Hinsehen entdeckte ich kurz hinter einem Wasserfall- und dem Spiralmodell meine Notizen zu *„Kapitel 3: Anforderungs- und Projektmanagement"*. Mit einem amüsierten Lächeln blätterte ich diese Seiten durch und dachte: „Früher war alles einfacher". Meine anhaltende Erheiterung galt dabei der Dokumentation einem meiner ersten Softwareprojekte. Die Semesterarbeit trug den Titel *„Medienverwaltung mit integriertem Leihsystem"* (MVLS). Damals hatten wir zu viert tagelang die Anforderungen akribisch aufgenommen, nach Wichtigkeit und technischer Machbarkeit sortiert und schließlich unserem Professor zur Korrektur vorgelegt, bevor wir mit dem für einen angehenden Softwareentwickler angenehmeren Teil anfangen durften. Lesen Sie im Artikel, mit welchen Hürden wir vier als Studenten unser Projekt abgeschlossen haben und was wir alle daraus heute noch lernen können.

Chaos

Natürlich hatte inzwischen der eine oder andere – also alle – von uns bereits heimlich angefangen zu programmieren.

Ein erster Schock: Viele der Überlegungen wurden von unserem Professor mit „irrelevant" und einige Anforderungen mit „nicht gewünscht" gekennzeichnet. Es kostete zwei Sprechstunden und einiges an Überzeugungsarbeit, um aufzuzeigen, dass einige Funktionen unbedingt notwendig sind und dass wir bei so wenig Inhalt mit der Software sowieso viel zu früh fertig sind. Als besonders strittiger Punkt erwies sich unsere „Mahnfunktion", deren Notwendigkeit der Professor als Auftraggeber offensichtlich zunächst nicht einsah.

Unser Konsens: Dem Auftraggeber war die prekäre Situation nicht bekannt, ein seltenes Buch innerhalb der Prüfungszeit ergattern zu können. Und wir wollten schließlich auch alle keine schlechten Noten, weil wir zu wenig getan hätten. Schließlich konnten wir uns doch noch bei der einen oder anderen – uns als sinnvoll erscheinenden – Funktion durchsetzen und sie mit in das Lastenheft aufnehmen.

Ordnung

Nachdem wir uns nach diesem anfänglichen Durcheinander auf einen ersten Weg geeinigt hatten, *was* wir eigentlich entwickeln wollten, kam die Idee auf, ein Requirements Management Tool zu nutzen. Alle bekannten Werkzeuge waren jedoch unhandlich und unkomfortabel. Sie

2.1) Medienverwaltung

In diesem Programmteil kann der Anwender den Medienbestand Verwalten, ihm stehen dafür folgende Möglichkeiten zur Verfügung:

- Medien anlegen
- Medien bearbeiten
- Medien löschen
- Medien suchen

Abb. 5: Kategorie „Medienverwaltung", Hauptansicht

```
====================MVLS====================
        -Medienverwaltung-
Bitte wählen sie einen Unterpunkt aus:

        1.) Medien anlegen
        2.) Medien bearbeiten
        3.) Medien löschen
        4.) Medien suchen

<Programm beenden>     <vorherige Seite>   <Hauptauswahl>

Eingabe:
```

Abb. 1: *Das Modul „Medienverwaltung" mit UI Prototype (Auszug aus der Semesterarbeit)*

waren weder intuitiv zu bedienen, noch konnten sie mit wirklich überzeugenden Funktionen glänzen.

Jeder halbwegs interessierte Datenbankentwickler konnte schließlich eine Access-Datenbank mit einem Eingabeformular entwickeln. Bei einigen Werkzeugen hatten wir sogar den Eindruck, dass selbst eine einfache Excel-Liste mehr Vorteile hätte.

Somit war auch die Entscheidung zum Requirements Management Tool gefallen. Eine Kombination aus Microsoft Office Word und einer einheitlichen Struktur sollte für unsere Zwecke genügen.

In **Abbildung 1** ist eine Eingangsanforderung „Medienverwaltung" lediglich grob umrissen dargestellt. Durch die Erweiterung um einen UI-Prototypen konnten wir uns vergewissern, dass alle Projektbeteiligten die gleiche Vorstellung über das Ergebnis haben. **Abbildung 2** zeigt eine andere, bereits weiter detaillierte Anforderung in der festgelegten, standardisierten Form mit Beschreibung, Eingaben und dem geschätzten Aufwand mit Risikobewertung.

Im Überblick: Die um einen UI-Prototypen erweiterte Kundenanforderung (oben) und detaillierte Ableitung der Umsetzung (unten).

An dem Anforderungsdokument ist auffällig, dass die Beschreibung stark aus der Sicht der Entwickler getrieben ist. Fast alle Anforderungen hatten folgende gemeinsame Punkte:

- Die Anforderungen sind in verständlichem Klartext und nicht in gestelzten Formulierungen gehalten.
- In der Beschreibung des Aufwandes wurden Risiken intuitiv vermerkt und beschrieben.
- Wegen fehlender Erfahrungswerte wurden die Aufwandsschätzungen oft relativ zueinander verknüpft (größer als …)
- Der Übergang von der Anforderung und der Vorstellung über die konkrete Art der Implementierung ist meistens fließend. Der Problembereich ist mit dem Lösungsbereich verknüpft.

Kommunikation

Die Implementierungsphase begann. Wir hatten uns nach Lehrbuch auf wöchentliche Statustreffen geeinigt: Jeden Montag nach der OOP-Vorlesung im Rauchereck (gab es damals noch) der Mensa. Eine demokratische Entscheidung vom Team. Der Plan war simpel: Projektbesprechung, Mittagessen und Rauchen zeitgleich. Das System kam nur schnell ins Wanken, wenn die Essensschlange in der Mensa wieder einmal zu lang war und einer der Teilnehmer vielleicht auch noch früher gehen musste.

In der Tat fühlten sich die Treffen eher gekünstelt und erzwungen an, und produktiv waren sie in den seltensten Fällen. Aber diese Termine waren auch mehr für das Papier und die Note. Die wirklich wichtigen Themen und die eigentliche Teamkommunikation fand *wann immer nötig* statt – über ICQ und E-Mail oder einfach zwischen den Vorlesungen am Kaffeeautomaten.

Gut eingebettet zwischen dem täglichen Social Talk fand man die notwendigen Informationen, um weiterarbeiten zu können: ein kurzer Schlagabtausch zu verketteten Listen, Wochenendplanung, Datenbank-Schema und der neuen Mitbewohnerin in der WG. Aus diesen Erfahrungen können wir folgende Leitlinien der Teamkommunikation für die Praxis ableiten:

- Über 70 % der Projekt- und Team-Kommunikation findet außerhalb geplanter Meetings statt.
- Reduzieren Sie die Anzahl der regelmäßigen Pflicht-Meetings auf ein Minimum und sorgen Sie dadurch für weniger Zeitvergeudung und Unterbrechungen.
- Spricht das Team außerhalb der Pflicht-Meetings nicht oder nur wenig miteinander und über das Projekt, kann es sein, dass es sich nicht mit dem Produkt identifiziert. Sie könnten ein Problem bekommen.

1.) Bücherverwaltung
- *a. Verleih*
- *b. Rückgabe*
- *c. Mahnbescheid*
  - *i. Beschreibung*
    1. *Das System soll täglich eine Liste der verliehenen und überfälligen Bücher generieren. Die Liste soll dem Benutzer nach dem Login angezeigt werden. Sie soll automatisch gedruckt werden, wenn sich kein Benutzer anmeldet. (Windows Scheduler? + Standard-Drucker)*
  - *ii. Benutzereingaben*
    1. *<todo> Hier noch die Benutzer-Dialoge einfügen*
  - *iii. Aufwand*
    1. *Größer als „Verleih" (wir brauchen einen Scheduler)*
    2. *Weniger riskant, wenn der Scheduler nicht funktioniert, kann die Funktion noch manuell ausgeführt werden*

Abb. 2: *Anforderung zum Modul „Mahnbescheid" (Auszug aus der Semesterarbeit)*

Implementierung

Schnell rächte sich, dass wir bereits vor dem Verabschieden des Lastenhefts mit ein paar Funktionen begonnen hatten. Es war für einen Entwickler deprimierend, wenn er auf das falsche Pferd gesetzt hatte und seinen Code wegen fehlender Anforderung verwerfen musste. Statt in guter alter Jean-Pütz-Manier mit „Ich hab da mal was vorbereitet" aufzuwarten, hieß es oft, die Code-Passagen aus dem eigentlichen Projekt zu löschen.

So wanderten unter anderem viele Zeilen Quelltext zum Erstellen und Drucken eines Mahnbescheids äußerst revisionsunsicher in den Papierkorb. Wir wussten wohl alle, was ein Versionsverwaltungssystem ist; und auch warum es sinnvoll wäre, dieses zu verwenden; und dennoch haben wir uns aufgrund des Projektes dagegen entschieden. Der Overhead, um ein solches System aufzusetzen, erschien uns im Vergleich zur Größe des Projektes ungerechtfertigt.

Wir bewerteten die bereits entstandenen Funktionen nach verschiedenen Kriterien:

- Brauchen wir diese Funktion wirklich? Ist sie gefordert?
- Will *ich* diese Funktion aus persönlichen Motiven?
- Kann diese Funktion dabei helfen, unser Verständnis zu verbessern? Gibt es einen Lerneffekt?
- Können wir diese Funktion später vielleicht noch einmal verwenden? Lohnt es sich hier mehr Zeit zu investieren?

Testen

Das Kapitel Testen war zügig erledigt. Nachdem die Daten auf einem Rechner manuell zusammengeführt wurden, wurde

feierlich F5 zur Ausführung des Programms gedrückt. Nach ein paar Warnungen, die ignoriert werden konnten – für die meisten Entwickler erlangen nur Fehler ihre Aufmerksamkeit –, wurden die Hauptfunktionen einmalig manuell ausgeführt: Test bestanden.

Da das Thema „Software Testing" den Rahmen bei Weitem sprengen würde, sei hier nur auf die einschlägige Literatur verwiesen (vgl. z. B. [Osh09] und [ISTQB]) und angemerkt:

- Testen Sie so früh wie möglich und so oft wie nötig.
- Testen ist elementarer Bestandteil der Softwareentwicklung.
- Entwickler testen im Normalfall ihren eigenen Quellcode nicht.

Auslieferung

Wir freuten uns: Einige Wochen lang hatten wir nun nichts von unserem Professor und Auftraggeber gehört und wir konnten in Ruhe unsere Software entwickeln, erweitern, „testen" und schließlich fertiggestellen. Und das „on time" (zumindest theoretisch).

Der verräterische Zeitstempel der finalen Version: *4:55 am*. Aus Zeitnot hatten wir in den letzten Tagen Nachtschichten eingelegt, um zumindest alle Funktionen rudimentär zu implementieren. Hinter einigen Menüeinträgen versteckten sich sogar lediglich UI-Mocks, in der Hoffnung, es würde keinem auffallen.

Doch was war geschehen? Warum hatten wir plötzlich zu wenig Zeit für die verabschiedeten Aufgaben? Wir hatten doch extra mehr Features vom Auftraggeber verlangt und alles genau ausgerechnet. Ohne es zu wissen hatten wir eines der Grundgesetzte der Softwareentwicklung bewiesen:

„Work expands so as to fill the time available for its completion." (Parkinsons Law)

Eine unglückliche Kombination aus zwei Punkten war uns zum Verhängnis geworden: Zum einen, ist am Ende der Arbeit noch Zeit übrig, so wird die vorhandene Arbeit vergoldet. Es wurden bereits funktionierende Bereiche überarbeitet, Variablen umbenannt und Codekommentare der Rechtschreibprüfung unterzogen.

Zum anderen hat die *Abwesenheit unseres Auftraggebers* dazu geführt, dass keine weiteren neuen Aufgaben begonnen wurden. Auch wenn das Modul „Mahnbescheid" letztendlich im Anforderungsdokument aufgenommen wurde, dachte keiner mehr darüber nach. Eine Möglichkeit zur Verfolgung und Überwachung des Aufgabenstatus hätte geholfen.

Wir wussten, dass die Hauptpunkte der Bewertung für die Abnahme die Lauffähigkeit und die Übereinstimmung mit den Anforderungen sind. Und es kam, wie es kommen musste: Der Kunde wollte genau die Funktionen sehen, die entweder noch fehlerhaft oder teilimplementiert waren.

Geschickt zirkelte der „Chefentwickler" durch die schwierigen Bereiche und gelobte Besserung und Bugfixes für die nächste Version. Am Ende half alles nichts: Wir mussten vor der Präsentation einer Funktion gestehen, dass sie noch nicht verfügbar ist – es war die Mahnfunktion. Wir hatten sie übersehen.

- Auch wenn das Marketing besser im Anzug aussieht: Ihr Team kennt die Software am besten. Lassen Sie die Entwickler das Produkt dem Kunden präsentieren.
- Gleichen Sie den Umfang der erstellten Implementierung gegen die Anforderungen ab. Klingt trivial? Warum werden dann aber in fast allen Projekten Anforderungen schlicht „vergessen"?

Retrospektive

Eins vorweg: *Natürlich* war das Projekt als Ganzes gesehen erfolgreich. Es gab im Verlauf noch viele Höhen und noch viel mehr Tiefen. Zum Beispiel als das Team beschloss nur noch zu dritt weiter zu gehen, da das vierte Teammitglied keine Arbeit ablieferte.

Das Team hatte hierzu nach Rücksprache die Befugnis, sich selbst zu regulieren. Ein Umstand, den ich nur selten in „Real-World-Projekten" wieder vorgefunden habe. Was für uns normal war: Wer nicht teamfähig und auch nicht kritikfähig ist, der sollte das Team besser verlassen.

Ich selbst habe damals zum ersten Mal ein Projekttagebuch geschrieben und einige meiner Notizen ergaben erst Jahre später einen Sinn. Sie können nun vielleicht lächeln und sich denken „Klar, ein fiktives Studentenprojekt bekommt doch jeder hin!", doch haben wir bereits in unserer Studienzeit trotz mangelnder Erfahrung viele Punkte intuitiv richtig gemacht. Gerade einmal drei Monate in Teilzeit waren veranschlagt.

Die Entwicklung des MVLS kann immer wieder als Beispiel für den Verlauf eines Projektes verwendet werden. Lehnen Sie sich einmal zurück, und denken Sie an Ihre letzten Anforderungen.

- Waren die Anforderungen vollständig, natürlich und verständlich gehalten?
- Sind Sie sich wirklich sicher, dass alle Beteiligten genau das verstanden haben, was Sie auch meinten?
- Stimmte das Ergebnis mit Ihrer Vorstellung und den Anforderungen überein?
- Hatten Sie und das Entwicklerteam einen Konsens über Komplexität, Risiko und vor allem Aufwand der einzelnen Features?

Den Effekt von Parkinsons Law und „Fertigstellung auf den letzten Drücker" können Sie abschwächen, indem Sie häufiger kleine Teile des Produktes planen und umsetzen. Ein Beispiel liefern hierzu die kurzen Iterationen beziehungsweise Sprints bei agilen Methoden von üblicherweise weniger als vier Wochen.

Verwenden Sie die notwendigen Werkzeuge und Tools, um ein einfaches Arbeiten zu ermöglichen. Ein Requirements Management Tool ist mindestens ebenso wichtig, wie ein revisionssicheres Quellcodeverwaltungssystem. Denn: Eine Softwarelizenz mit entsprechender Hardware ist allemal günstiger als ein fehlgeschlagenes Projekt; vom psychologischen Faktor für die Projektbeteiligten mal ganz abgesehen.

Selten lassen sich diese beide Welten integrieren. Eine mögliche Lösung im Microsoft-Umfeld liefert die Kombination aus Microsofts Team Foundation Server (TFS) [TFS1] und der serverseitigen Erweiterung TFS ASAP [TFS2] an.

Durch konfigurierbare, automatisierte Dienste werden Sie darüber informiert, wenn sich Aufgaben oder ganze Anforderungsbäume (Work Breakdown Structures, WBS) ändern. Sie können somit jederzeit den aktuellen Status des Projektes beurteilen und zeitnah darauf reagieren. Auch wird der jeweilige Aufwand über die einzelnen Anforderungsebenen aggregiert – so bleiben Sie einfacher „on budget" und vermeiden eine ärgerliche Budgetüberziehung.

Fazit

Es tut mir leid, wenn ich Ihnen hier nicht der Weisheit letzten Schluss präsentieren

kann oder eine Philosophie, die Ihr Leben grundlegend verändert. Nur vielleicht so viel: Anforderungen werden heute oft als Vertrag angesehen. So wie Sie mit Ihrem Handwerker vereinbaren, welche Fliesen Sie im Bad auf dem Boden und welches Waschbecken an der Wand haben wollen.

Für Sie ist das Thema nach der Auftragsvergabe an den Handwerker aber sicher nicht abgeschlossen. Da Sie wissen, dass es am Ende problematisch wird, wenn die Fliesen quer statt diagonal verlegt sind, reden Sie vorher noch einmal mit dem Fliesenleger darüber. Auch machen Sie ihn mit dem Heizungsinstallateur bekannt und geben beiden noch einmal Ihre Nummer „falls Sie noch Fragen haben…".

Ähnlich verhält es sich mit den Anforderungen in der Softwareentwicklung, denn die damit verbundenen Tätigkeiten können nicht als alleinstehendes Element betrachtet werden. Viele Parteien wie Einkauf, Management, Lieferant(en) – und wenn Sie Glück haben vielleicht auch Entwickler – arbeiten an diesen Machwerken.

Alle Bereiche der Softwareentwicklung müssen als eng verzahntes Uhrwerk betrachtet werden. Doch längst ist eine Kultur entstanden, in der Anforderungen scheinbar ausschließlich zur Absicherung des Vertrages verkommen. Das Produkt selbst rückt derart oft in den Hintergrund.

Statt alles daran zu setzen, das Projekt erfolgreich abzuschließen, rüstet sich jede der beteiligten Parteien zuallererst für den Fall, dass das Projekt **scheitert**.

Herzlichen Glückwunsch: Thema verfehlt – setzen – 6. ■

Referenzen

[Osh09] (Zum Beispiel für den Entwickler) „The Art of Unit Testing" von Roy Osherove, erschienen 2009 im Manning Verlag.

[ISTQB] International Software Testing Qualification Board – ISTQB; http://www.istqb.org

[TFS1] Micosoft Team Foundation Server; http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx

[TFS2] TFS ASAP „Automated Servicing & Administration Platform"; http://www.tfsasap.com