> If you want to confuse your enemies, give them the source code. If you want to really confuse them, give them the documentation. - unknown

Wie lässt sich eine ausreichend gute Anforderungsspezifikation schreiben? Wie schaffen wir es das Wissen um die Fachlichkeit bestmöglich in Sourcecode umzuwandeln? Und wie kann ich, auch in agilen Projekten, den Fach- und Technikkollegen nach dem Projektende diese Fachlichkeit als Dokumentation zur Verfügung zu stellen?

Hindernisse bei der Dokumentation

Jira-Tickets, bunte Post-its funktionieren gut während der Anforderungsentdeckung und während der Umsetzung. Aber was ist mit der Zeit nach dem Projekt? Hunderte Seiten Microsoft Word mit Satzschablonen (Soll, Muss, Kann) hingegen sind weder während der Umsetzung noch danach der Stoff aus dem die Leserträume sind.

Schauen wir uns kurz die Definition von „Spezifikation“ und „Dokumentation“ an. Spezifikationen sind eine Auflistung von Anforderungen zum Zwecke der Kommunikation, um ein gemeinsames Verständnis des Produktes zu erlangen. Die Dokumentation ist die Nutzbarmachung von Informationen zur weiteren Verwendung. (Quelle Wikipedia).

Mit dem kecken Spruch der *Code ist die Dokumentation* sollten wir uns nicht zufriedengeben. Der Aspekt „nutzbar machen“ der Dokumentation ist bei Sourcecode nur bedingt gegeben, da die meisten Kollegen aus dem Fachbereich ihn nicht lesen können. Sie haben meist keinen Zugriff darauf und die wenigsten können Quellcode lesen/ verstehen. Stattdessen haben wir ein Monopol und einen Engpass bei der Technik geschaffen. Das Monopol gehört dort nicht hin, denn über die Fachlichkeit sollte die Fachseite die Hoheit haben. Den Engpass wollen wir nicht, denn wir binden kostbare Entwicklerzeit, wenn wir aufgrund „nicht lesbarer“ Dokumentation über Reverse-Engineering die Fachlichkeit wieder aus dem Code extrahieren müssen. Das ist langwierig, schmerzhaft und teuer.

User Stories sind zwar ein gutes Mittel, die Nutzerwünsche zu entdecken und eignen sich hervorragend als Startpunkt für die Kommunikation zwischen Fach- und Technikteams. Sie sind aufgrund ihrer Natürlichsprachigkeit für alle Projektbeteiligten lesbar und leicht verständlich. Für die Dokumentation nach dem Projekt sind sie, für sich allein, eher ungeeignet, es fehlt das Fleisch der Anforderungen am Knochen der User Story“.

Die klassischen Anforderungen in epischen Ausführungen mit Satzschablonen sollten wir weder unseren Kollegen in der Technik anbieten, noch sind sie als Dokumentation für den Fachbereich geeignet. Bei aller Standardisierung (soll, muss, kann, etc.) bieten auch die Satzschablonen Interpretationsspielraum bzw. sind umständlich zu lesen. Ein ganzheitliches und gemeinsames Verständnis zwischen Ersteller und Leser schaffen sie auch nur schwer. Noch schlimmer wird es nur, wenn die in der Analysephase erstellten Anforderungen nicht während der Umsetzung aktualisiert wurden. Dann ist die Dokumentation nicht nur schwer lesbar, sondern auch veraltet und stimmt natürlich nicht mit der Fachlichkeit im Quellcode überein.

Warum nicht das Entdecken der Anforderungen in der Konzeptionsphase mit der Spezifikation für die Umsetzung und Dokumentation für danach verbinden? Der Charme liegt auch darin, dass die „Spezifikation“ als „Dokumentation“ nach einem Projekt genutzt werden kann, da die Fachlichkeit und die Anforderungen sichtbar und lesbar sind. Eine gute Spezifikation und Dokumentation ist besonders wichtig, wenn Software durch externe IT-Dienstleister erstellt wird und der Zugriff auf den Quellecode dann ggf. nur sehr schwer möglich ist oder gar unerwünscht ist.

Wie Spezifikation zur Dokumentation wird

Das Bindeglied können *Beispiele* und daraus abgeleitete Testfälle sein, diese helfen die Brücke zwischen Business und Technik zu schlagen. Sie stellen die fachlichen Wünsche/ Anforderungen für das gewünschtes Systemverhalten für alle Projektbeteiligten plastisch dar. So lassen sich Missverständnisse während der Konzeption aus dem Weg räumen und in den Beispielen und Testfällen sind die Geschäftsregeln gut dokumentiert. [1] [2]

Die Beispiele sind die „Anforderungen“ mit denen Entwicklungsteams die Anwendung bauen und die von der QA/ QS genutzt wird, um die Software zu prüfen. Hierbei gibt es durch die Beispiele weniger Interpretationsspielraum als bei „normalen“ Anforderungen und im Entwicklungsprozess werden wir angeregt an weitere Beispiele/ Testfälle und Produktoptionen zu denken.

Durch die gemeinsame Entwicklung (Fach- und Technikbereich) von Beispielen vor der eigentlichen Umsetzung werden außerdem Inkonsistenzen der Nutzer- oder Systemanforderungen deutlich und bisher nicht berücksichtigte Geschäftsregeln aufgedeckt. Systemverhalten wird quasi auf dem Whiteboard durchgespielt. Wir investieren nicht erst Entwicklungszeit, um dann im Test herauszufinden, dass dies nicht dem gewünschten Systemverhalten entspricht.

Beispiel:
Nehmen wir an, wir entwickeln einen Shop, bei dem ein Kunde ein Guthaben aufladen und dann von diesem Guthaben Waren einkaufen kann.

Vom Produktmanagement gibt es einen schnellen Zweizeiler Richtung IT Abteilung:

> *Der Kauf einer Ware soll möglich sein, wenn das Guthaben größer als der bestellte Warenwert ist.*

Die Anforderung scheint eindeutig und richtig, oder? Erinnern wir uns aber kurz an die Zeiten von Kaufmannsläden, dann gab es die begehrte Tüte Bonbons auch, wenn uns ein Pfennig gefehlt hat, oder?

In einem Miniworkshop mit der Technik und der Fachseite wird die Anforderung nochmal betrachtet. Das Technikteam hat sich dafür ein paar Testfälle überlegt und präsentiert diese dem Produktmanagement.

Mögliche Beispiele laut oben beschriebener Anforderung können wie folgt aussehen:

1. *Guthaben 10 Euro - Warenwert 8 Euro = Verkauf OK*
2. *Guthaben 10 Euro - Warenwert 15 Euro = Verkauf nicht OK*
3. *Guthaben 10 Euro - Warenwert 9,99 Euro = Verkauf OK*
4. *Guthaben 10 Euro - Warenwert 10,01 = Verkauf nicht OK*

Spätestens beim Testfall 4 sollten die Kollegen des Vertriebs aufspringen und ihr Veto einlegen.

Sollen wir auf Umsatz, Kundenzufriedenheit und 4 Euro Marge verzichten, nur, weil wir eine Unterdeckung von einem Cent haben? Besonders vor dem Hintergrund, da das Geschäftsmodell auf Bestandskunden basiert, die ihr Guthaben häufig aufladen, um damit einzukaufen.

In der Diskussion mit dem Team wurden verschiedene Optionen besprochen wie mit Beispiel 4 umgegangen werden kann.

A. Kein Verkauf, Guthaben muss größer sein als der gewünschte Warenwert;
B. Aufladen des Guthaben während des Zahlungsprozesses mit anderer Zahlungsoption;
C. Zahlung des Differenzbetrages mit anderen Zahlungsoptionen;
D. Kreditrahmen;
E. Verkauf mit Kreditrahmen und Start eines Folgeprozesses „Guthaben aufladen“.

Nach Einschätzung der Technik wäre Punkt A und D innerhalb des Sprints umsetzbar. Da der Produktmanager aber den Umsatz nicht verlieren möchte und seinen Stammkunden ein gewisses Vertrauen entgegenbringt entscheidet er sich für die Option D, „Kreditrahmen“

Das Team (Technik und Fachseite) nähert sich jetzt der zumutbaren Grenze mit Beispielen an.

5. *Guthaben 10 Euro - Warenwert 10,01 = Verkauf OK*
6. *Guthaben 10 Euro - Warenwert 10,20 = Verkauf OK*
7. *…*

Am Ende einigt man sich auf einen „Kreditrahmen“ von 35 Cent bzw. auf einen Prozentrahmen[1] vom Warenwert.

8. *Guthaben 10 Euro - Warenwert 10,35 (3,5%) = Verkauf OK*
9. *Guthaben 10 Euro - Warenwert 10,36 (3,6%) = Verkauf nicht OK*

Nach dem man sich geeinigt hat, bringt ein Entwickler die Frage auf, was passiert, wenn der Kunde ein negatives Guthaben hat. Wir müssen sicherstellen, dass dem Kunden sein korrektes Guthaben angezeigt wird. Im Falle eines Kreditrahmens müsste das dann ein negatives Guthaben sein. Daraus ergeben sich weitere Anforderungen und Testfälle.

Nach der Runde mit der Technik ergibt sich ein klares Bild über die gewünschten Funktionen und die Spezifikation für die User-Story kann präzisiert werden. Hierfür werden die Lösungsoptionen für unsere User-Story definiert und zwar in den Dimensionen („*User*“, „*Interface*“, „*Action*“, „*Data*“, „*Control*“, „*Environment*“ und „Quality“) [4]. Neben den Testfällen reichert das Projektteam für den nächsten Sprint die User-Story noch mit ein paar technischen Details an. Der gewünschte Aufwand ist im nächsten Sprint umsetzbar. Für das Entwicklerteam kann die Anforderung wie folgt aussehen:

Als Guthabenkunde möchte ich trotz unzureichendem Guthaben einkaufen können, damit ich meine gewünschten Waren/Dienstleistungen schnell nutzen kann.

Anforderungen:
User:
- gilt nur für Guthabenkunden

Data:
1. akt. Guthabendaten
2. akt. Warenwert im Warenkorb

Interfaces (Maschine-Maschine & Mensch-Maschine):
1. Domäne „Guthabenverwaltung“
2. Domäne „Warenwirtschaft“
3. Domäne „Kundenkommunikation“
4. GUI Mock up für die Fehlermeldung

[1] Gerundet auf die erste Nachkommastelle

Action:
1. Kauf von Waren mit Kreditrahmen
2. Anzeige von positiven Guthaben
3. Anzeige von „negativen“ Guthaben (Kreditrahmen)
4. Versand der E-Mail „Erinnerung zur Kontoaufladung“

Control:
1. Gewährung eines Kreditrahmen von bis zu 3.5% des Warenwertes
2. E-Mail „Erinnerung zur Kontoaufladung“ an Kunden senden

Environment
1. Desktop, Firefox und Safari

Quality
1. 24/7 Erreichbarkeit
2. E-Mail „Erinnerung zur Kontoaufladung“ wird ca. 1h später geschickt

Die erarbeiteten Testfälle werden als Akzeptanzkriterien zur User Story hinzugefügt. Für den Sprint gibt es noch weitere Informationen hinsichtlich der nötigen Kunden und Verkaufsdaten, sowie ein Mockup als Beispiel für die Nutzeroberfläche.

Testfälle könnten unter anderem wie folgt aussehen:

1. Vorausgesetzt, ein Kunde zahlt mit Guthaben
    a. Wenn der Warenwert 10,35 € und der Guthabenwert 10 € ist
        i. Dann kann der Kauf vollzogen werden
2. Vorausgesetzt, Kunde zahlt mit Guthaben
    a. Wenn der Warenwert 10,36 € und der Guthabenwert 10,36 € ist
        i. Dann ist der Verkauf nicht möglich
        ii. Dann soll folgender Fehlertext angezeigt werden: „Guthaben für den Kauf nicht ausreichend, bitte Guthaben aufladen oder Warenwert reduzieren.“
3. Vorausgesetzt, ein Kunde schaut sich den aktuellen Guthabenstand an
    a. Wenn das Guthaben positiv ist
        i. Dann wird der Wert im folgendem Format (Farbe Schwarz) angezeigt:
            1,00 / 123, 57 / 77, 65
    b. Wenn das Guthaben negativ ist
        i. Dann wird der Wert im folgenden Format (Farbe Rot) angezeigt:
            0,67 / 1,24 / 10, 58 / 123,00
4. Vorausgesetzt, ein Guthabenkunde schließt einen Bestellprozess ab
    a. Wenn das Guthaben im Anschluss des Bezahlvorgangs negativ ist
        i. Dann versende Erinnerungsmail „Erinnerung zur Kontoaufladung“ an hinterlegte E-Mail-Adresse des Kunden
5. Vorausgesetzt, …

Wie wir sehen konzentrieren wir uns auch nur auf die Zahlung mit Guthaben. Die anderen Zahlungsweisen werden ausgeklammert und ggf. in einer anderen User Story bzw. in einem späteren Sprint betrachtet. Der Testfall und die Dokumentation könnten dafür wie folgt aussehen.

Als Kreditkartenkunde möchte ich mit Visa einkaufen können, damit ich meine gewünschten Waren/Dienstleistungen schnell nutzen kann.

Testfälle könnten unter anderem wie folgt aussehen:

1. Vorausgesetzt, ein Kunde zahlt mit Visa
    a. Wenn der Zahlungsanbieter die Deckung des Rechnungsbetrag nicht bestätigt
        i. Dann ist der Verkauf nicht möglich
        ii. Dann soll folgender Fehlertext angezeigt werden: „Kreditkartenanbieter autorisiert Zahlung nicht“
2. Vorausgesetzt, …

Zusammenfassung

Beispiele und Testfälle eignen sich hervorragend für die Diskussion in der Konzeptions- und Umsetzungsphase. Sie schaffen ein gemeinsames Verständnis über das Systemverhalten und zwar bereits vor der Entwicklung. Diese Arbeit führen wir auch in agilen Projekten durch. Da sie Natürlichsprachig gehalten sind können sie nach dem Projekt auch für die Dokumentation der Fachlichkeit des Systems genutzt werden. Den Kritikpunkt, dass agile Projekte keine Dokumentation erstellen kann so leicht entgegengewirkt werden. Natürlich sollte die Dokumentation dann für alle zugänglich sein, sowohl Fachbereichen als auch Technikteams. Die Aktualisierung ist dann in der Verantwortung der Fachbereiche.

Fachliche Anforderungen, sollten von der Fachseite verantwortet werden. Damit diese aber die Verantwortung übernehmen kann, muss die Fachseite die Anforderungen lesen und verstehen können. Sourcecode können sie meist nicht lesen.

Happy Engineering
René

www.requireminds.de

Quellen:
[1] Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing
[2] Specification by Example: How Successful Teams Deliver the Right Software
[3] Wikipedia (Def. von Spezifikation und Dokumentation)
[4] Discover to Deliver - Agile Product Planning and Analysis