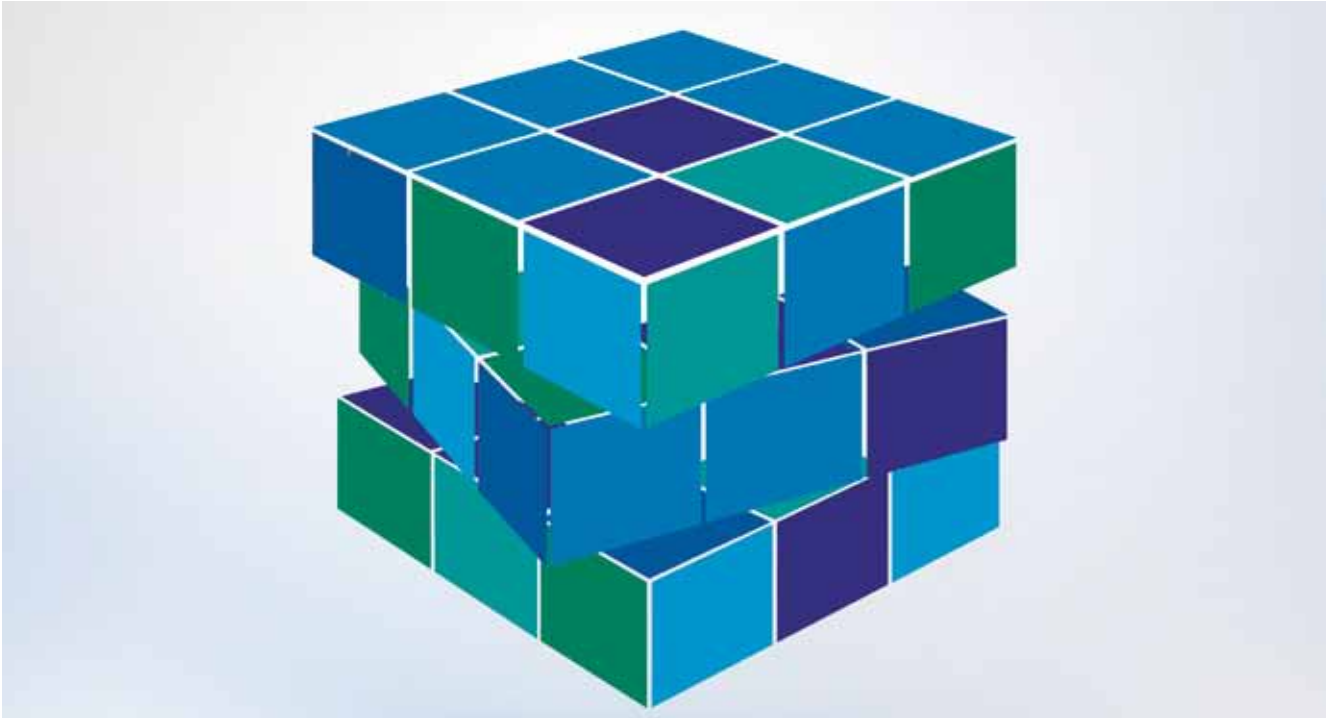


Photo: Shutterstock / thesomeday123

SOFTWARESANIERUNG, ABER RICHTIG

Auf den Putz hauen

Die Entwicklung von Features dauert immer länger? Die Software aufzuräumen erlaubt der Chef aber nicht? Vorsicht und die richtigen Argumente führen zum Ziel!

Als erfahrener Entwickler kennen Sie das sicher: Mit Anlauf und voller Elan springt man in eine neue Herausforderung, nur um sich nach langem Flug die Nase an den Härten der Realität aufzuschlagen. Dies kann in vielen verschiedenen Situationen wie beim Refaktorisieren von Code oder der Einführung neuer Frameworks geschehen. Was gerade noch aussah wie eine verhältnismäßig leicht zu meisternde Aufgabe, kann unerwartet zum unüberbrückbaren Hindernis mit Chaospotenzial werden, welches in aller Regel auch noch neben dem sonst schon anstrengenden Alltagsgeschäft passieren muss.

All dies kann dann nur noch dadurch getoppt werden, dass die Aufgabe noch größer, die Auswirkungen noch schwerwiegender und der Druck noch höher ist, so zum Beispiel bei der Sanierung oder Restrukturierung eines bereits bestehenden Systems. Gerade in einem solchen Umfeld ist die Planung meist sehr schwierig und das sichtbare Ergebnis nur schwer zu verkaufen.

Dieser Artikel will Ihnen dabei helfen, die richtigen Schwerpunkte zu setzen und nicht zu unbedarft vorzugehen. Der Artikel zeigt zu diesem Zweck die Herausforderungen außerhalb der reinen Softwareentwicklung und versucht eine Hilfestellung anzubieten, wie man diesen begegnen kann, insbesondere bei der Pflege bestehender Software.

Softwaresanierung

Jede Software unterliegt einem Lebenszyklus mit unterschiedlichen Phasen und Arbeitsschwerpunkten. Bestimmt zu Beginn noch die Umsetzung neuer Features den Arbeitsalltag, werden nach einigen Jahren Dinge wie Optimierung und Stabilisierung wichtiger (Bild 1).

Damit verschiebt sich der Fokus auch zunehmend weg von der Steigerung des Wertes der Software in Form von neuen Features hin zur Werterhaltung durch Anpassungen von bestehender Funktionalität. Dies steigert sich bis zu dem Punkt, an dem Bestandteile migriert oder ausgetauscht werden müssen, um die Software arbeitsfähig zu halten. Spätestens an diesem Punkt sprechen wir von Sanierungsaufgaben. Dabei wird eventuell das User Interface (UI) ausgetauscht oder zentrale Komponenten neu strukturiert. Gerade Letzteres bietet ein gewisses Konfliktpotenzial, da die damit verbundenen Anpassungen für den Benutzer möglicherweise keine sichtbaren Verbesserungen bieten, zeitgleich aber Arbeitsleistung binden, die an anderer Stelle fehlt.

Gerade Entscheider ohne ausgiebige Programmiererfahrung neigen deshalb dazu, reinen Umstrukturierungen kritisch gegenüberzustehen, weil sie den Wert der Anpassbarkeit von Software nur schwer ermessen können. Ihre Referenzgröße für Mehrwert ist demnach vorrangig das gebotene

Feature Set, und dieses kann durch Aufräum- und Umbauarbeiten eben auch schrumpfen.

Geschäftswert versus Anpassbarkeit

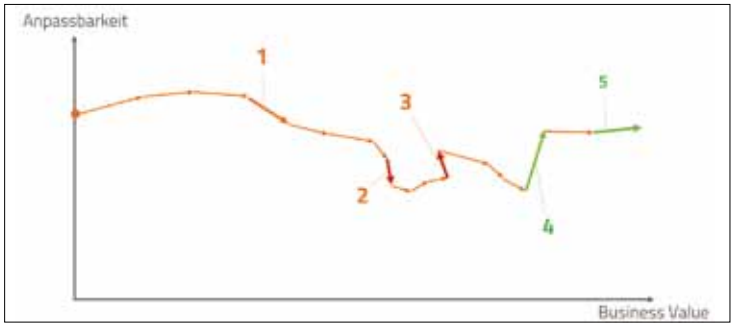
Bild 2 illustriert den Zusammenhang zwischen Geschäftswert und Anpassbarkeit. Während die interne Qualität der Software zu Beginn noch recht hoch ist, liegt der Geschäftswert bei 0, immerhin bietet die Software ja keinerlei relevante Funktionalität. Mit fortschreitender Entwicklung wächst die Funktionalität und damit auch der Wert ihres Einsatzes. Sollte bei der Implementierung aber nicht konstant auf die innere Qualität geachtet werden, wozu das Schreiben automatisierter Tests und das Refaktorisieren ungünstig gewählter Strukturen zählt, erschwert dies die zukünftigen Umsetzungen neuer Funktionalität und macht sie damit teurer (Markierung 1 in Bild 2). Verschlimmert wird dies noch durch Implementierungen ohne jede Rücksicht auf die internen Strukturen, wie es beispielsweise unter hohem Zeitdruck geschieht. Dabei werden gern einfach Codeblöcke kopiert und somit Codeduplikate erzeugt, welche ihrerseits noch viele weitere Probleme nach sich ziehen. Die Software erleidet in diesem Moment einen erheblichen Wertverlust im Bereich der Anpassbarkeit, den Benutzer und andere Außenstehende meist erst dann wahrnehmen, wenn er sich bis hin zu Fehlern im Produktivbetrieb durchschlägt (Markierung 2 in Bild 2).

Nicht nur, dass die erbrachte Leistung kaum Mehrwert für den Benutzer gebracht hat, sie hat eventuell sogar die Arbeit von Nutzern und Entwicklern behindert, weshalb der betroffene Teil schleunigst in Ordnung gebracht werden sollte. Unter Umständen kann es dann als gute Idee erscheinen, den Code auszubauen und damit das Feature Set der Software zu verringern (Markierung 3 in Bild 2). Diese Arbeiten werden wiederum von den Auftraggebern ungern gesehen, da sie bereits getätigte Investitionen zunichte machen, bedeutet ja ein Entfernen der Features, dass das ursprünglich aufgewendete Budget falsch investiert wurde, neue Investitionen für den Ausbau vorgenommen werden müssen und die Benutzer am Ende dafür sogar noch weniger Leistung bekommen. Änderungen dieser Art müssen daher sehr gut abgesprochen und begründet werden. Dies verlangt von uns Technikern ein gewisses Fingerspitzengefühl in der Kommunikation technischer Zusammenhänge, der man sich nicht unvorbereitet stellen sollte. Ein besseres Vorgehen besteht darin, Anpassungen an der Struktur immer auch an neue Funktionalität zu koppeln und die so entstehenden Sanierungsaufwände in den Umsetzungsaufwänden des neuen Features aufgehen zu lassen (Markierung 4 in Bild 2).

Arbeitsschwerpunkte in der Software-entwicklung (Bild 1)

frei nach Tom Gilb, Evolutionary Delivery vs Waterfall Model, ACM Software Engineering Notes, Juli 1985, https://dl.acm.org/citation.cfm?id=1012490

Sollte also beispielsweise die verwendete Reporting-Komponente veraltet sein, so spricht man mit dem Fachbereich den Austausch auf Basis neuerer Reports oder anhand von Änderungen an bestehenden Reports ab. In dem Moment wird zwar die Umsetzung der Reports teurer, als sie eigentlich wäre, wenn man nur die Anpassung umsetzen würde, der Fachbereich hat aber am Ende auch neue Funktionalität und kann die Anpassungen auch anhand der neuen Anforderungen testen. Es empfiehlt sich in jedem Fall nicht, die Anpassung ohne Kommunikation der Umstände vorzunehmen. Da Umstrukturierungen dieser Art meist sehr weiträumige Codebereiche betreffen, besteht auch immer die Gefahr von Fehlern. Werden die Benutzer beziehungsweise Fachbereiche davon unvorbereitet getroffen, können Probleme erwachsen, die sich sehr negativ auf die Leistungsfähigkeit der Nutzer und das Vertrauen in die Software auswirken. Als Resultat ist dann häufig zu beobachten, dass eine generelle Skepsis gegenüber Umbaumaßnahmen vorherrscht und jene somit noch schwerer zu verteidigen sind. Im Beispielszenario kann es somit sein, dass das „Verstecken" von Sanierungsmaßnahmen bei der Umsetzung den Report unglaublich teuer macht und die Umstrukturierung gegebenenfalls mit den falschen Zielen vorgenommen wird. Klärt man hingegen proaktiv mit allen Stakeholdern, dass die Reporting Engine umgebaut werden muss, können mehrere aufeinanderfolgende Reporting-Sprints abgesprochen werden, in denen sich nur den Reports und den damit verbundenen Komponenten zugewendet wird. Auf diese Weise bleiben die Sanierungskosten im Rahmen, da sie über ein größeres Feature Set verteilt werden können. Weiterhin verringert die Konzentration auf einen Themenbereich (technisch und/oder fachlich) die Reibungsverluste im Team, da sich dieses nicht immer wieder neu mit der Thematik auseinandersetzen muss. Außerdem stellt man somit auch sicher, dass sich die Umbaumaßnahmen wirklich an den tatsächlichen Anforderungen der Benutzer orientieren und nicht an den vermeintlichen, die die Entwickler aus dem bestehenden Quellcode herausgelesen haben.

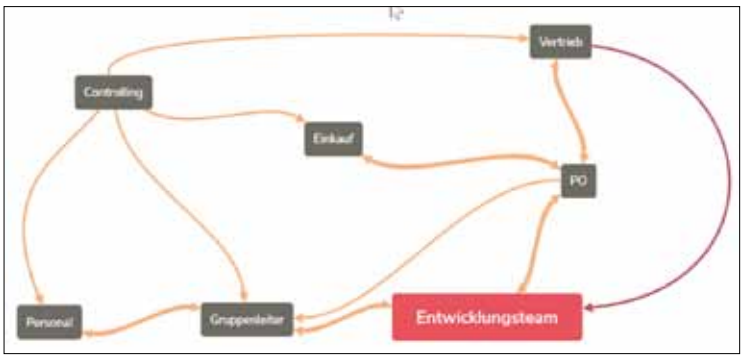


Anpassbarkeit versus Business Value (Bild 2)

frei nach Stephan Murer, Bruno Bonati and Frank J. Furrer, Managed Evolution, A Strategy for Very Large Information Systems, Springer-Verlag Berlin

Es mag wie eine Binsenweisheit klingen, aber die beste Möglichkeit, diese Sanierungsaufwände zu verringern, besteht darin, die notwendigen Absprachen und Planungen im generel- ▶

len Entwicklungsprozess und nicht als einmalige Aktionen unter erheblichem Leistungseinsatz durchzuführen. Natürlich lassen sich Umbaumaßnahmen nicht vermeiden, da auch bestehende Programmteile in größerem Maße an neue Gegebenheiten angepasst werden müssen. Sie fallen jedoch geringer aus, wenn man werterhaltende Maßnahmen von Beginn an in die Umsetzung von Features einpreist, sich der Pfadfinderregel bedient und auf eine ausreichend große Testabdeckung achtet (Markierung 5 in Bild 2). Als Daumenregel ist es sinnvoll, je nach Alter und Qualität des Quellcodes etwa zwanzig bis dreißig Prozent der Arbeitszeit für Maßnahmen aufzuwenden, die der Anpassbarkeit dienen. Dazu gehört dann aber ebenfalls nicht nur die reine Umsetzung dieser Maßnahmen, sondern auch deren Überprüfung mit Codereviews, statischer Codeanalyse und so weiter. Denn letztendlich reicht es nicht, den Entwicklern einfach nur mehr Zeit zu geben. Diese Zeit wird in jedem Fall genutzt werden, ob sie hingegen sinnvoll genutzt wird, kann man nur einschätzen, wenn man sie mit messbaren Zielen verbindet und die Zielerreichung regelmäßig prüft.



Fiktive Stakeholder Map (Bild 3)

Kenne deine Stakeholder

Oft genug fürchten sich Entwickler jedoch davor, qualitätssteigernde Maßnahmen implizit in die Aufwände von Features einzurechnen, und antworten auf die Frage „Wie lange brauchst du für die Umsetzung?“ mit einer Zahl, die nur den Teil betrifft, der nötig ist, um die grundsätzlichen Arbeiten abzuschließen. Oft genug machen sie den Fehler, das Dokumentieren, Testen und Aufräumen als weitere Einzelposten bei der Schätzung aufzulisten. Dies suggeriert aber wiederum, dass sie optional und damit Teil einer Verhandlungsmasse wären. Im schlimmsten Fall sind die Budgetverantwortlichen dann aber so sehr an Verhandlungen gewöhnt, dass sie den vermeintlichen Köder schlucken und probieren, für sich ein günstigeres Angebot herauszuholen, indem sie die werterhaltenden Maßnahmen wegdiskutieren.

Kommt nun noch hinzu, dass der entsprechende Entwickler ohnehin keine Lust auf das Schreiben von Tests und Dokumentation hat, dann beginnt eine Abwärtsspirale, die auf lange Sicht in einem riesigen Code-Klumpen endet, den niemand mehr anfassen möchte.

Wer also die Software verbessern möchte, muss ganz genau wissen, mit wem er es zu tun hat und wie er mit den entsprechenden Personen kommunizieren sollte. Daher sollte man sich fragen:

- Wer kann und sollte welche Aussagen treffen?
- Wer kann und sollte welche Entscheidungen treffen?
- Welche Motivationen haben die einzelnen Gruppen beziehungsweise Personen?
- Wer sollte mit wem reden und wer sollte es besser nicht?
- Wie muss man mit den jeweiligen Personen sprechen, um Informationen effektiv zu vermitteln?

Das klingt eventuell etwas verschwörerisch, aber tatsächlich sind Probleme im Quellcode oft genug nur Symptome für einen schlechten Softwareentwicklungsprozess und schlechte Kommunikation. Beseitigt man die Probleme im Quellcode, ohne die Probleme in der Zusammenarbeit zu beseitigen, werden die Probleme im Code immer wieder auftreten.

Die Erkenntnisse der genannten Fragestellungen können in eine sogenannte Stakeholder Map eingetragen werden. Dabei lohnt es sich, zwischen einer Ist- und einer Soll-Version der Karte zu unterscheiden. Dies hilft dabei, sich selbst der Problemstellen besser bewusst zu werden und diese auch anderen Personen zu erläutern.

Das fiktive Beispiel aus Bild 3 zeigt diverse Zusammenhänge auf. So hat das Entwicklungsteam zwar einen Product Owner (PO), der Anforderungen des Einkaufs und Vertriebs zusammenfasst, der Vertrieb neigt aber dazu, diesen zu umgehen und sich mit Anfragen direkt an einzelne Entwickler zu wenden. Dieser Kurzschluss der Kommunikation sorgt dann dafür, dass die Planung immer wieder durcheinandergerät, weil mitten im Sprint neue Anforderungen auftauchen, die andere verdrängen. Solche Kurzschlüsse sind unbedingt zu vermeiden. Sie stören nicht nur den Arbeitsablauf des Teams, sondern können auch dazu führen, dass sich der Einkauf benachteiligt fühlt und irgendwann selbst zu unfairen Mitteln greift, um die eigenen Ziele zu erreichen. In diesem Fall verkommt der Product Owner dann zu einer reinen Sockenpuppe, die eigentlich keinen Einfluss mehr auf die zu erledigenden Aufgaben hat. Symptome dafür sind aber nicht nur die nachträglich in den Sprint injizierten Aufgaben, man merkt diesen Umstand auch daran, dass die Aufgabenpakete sehr kleinteilig und ohne größeren Zusammenhang sind. Die Vision, die der Product Owner eigentlich für das Produkt haben muss, geht verloren und die Arbeit gleicht eher einer Flickschusterei als einer gezielten Entwicklung.

Kommunikationsschwierigkeiten

Angenommen, das Kind ist aber schon in den Brunnen gefallen und wir wollen nun den Code verbessern, wie hilft uns da die Stakeholder Map? Es liegt in der Natur der Sache, dass wir immer den Dingen eine höhere Priorität zumessen, mit denen wir am häufigsten zu tun haben. Es ist nur natürlich, dass für uns als Architekten und Softwareentwickler die Anpassbarkeit von Software ein höheres Gut ist, als sie es beispielsweise für den Benutzer ist, und es ist auch unsere Aufgabe, diese zu wahren, da andere Projektbeteiligte nicht den

Einblick in die Software haben können, den wir haben. Sie beauftragen uns ja gerade damit, die Dinge am Laufen zu halten, so wie wir als Privatpersonen beispielsweise den Schornsteinfeger oder Hausmeister mit der Wartung unserer heimischen Heizanlage beauftragen. Von diesem erwarten wir auch, dass er uns klar verständlich macht, wenn etwas nicht geht oder sich zukünftig negativ auswirken kann. Er kann seinerseits aber nur Vorschläge zur Verbesserung machen, und wir müssen auf Basis einer Mischung aus unvollständigen Informationen, Bauchgefühl und Budgetverantwortlichkeit dann die vermeintlich richtige Entscheidung treffen. In unserem Entwickleralltag sind wir aber nun einmal die Schornsteinfeger und Hausmeister, wodurch wir nicht frei entscheiden können. Wir dürfen uns nicht nur auf unsere Probleme, unsere Sichtweise und unsere Lösungen konzentrieren, sondern müssen uns in die Rolle all derer versetzen, die von den Auswirkungen unserer Arbeit betroffen sein können.

Die Stakeholder Map hilft uns dabei, mit den richtigen Personen in Kontakt zu treten, um mit ihnen zu klären, was aus ihrer Sicht ernsthafte Schwierigkeiten sind, was sie behindert und wo sie akuten Handlungsbedarf sehen. Dabei ist es wichtig, die Erfassung der Problemstellen zunächst als das anzusprechen, was sie zu dem Zeitpunkt ist: eine Bestandsaufnahme, bei der man noch nicht über konkrete Gegenmaßnahmen spricht. Neben der reinen Nennung von Problemen ist dabei auch wichtig, die Motivation der Gesprächsteilnehmer zu klären, um die Fortschritte der Arbeiten später besser darstellen zu können. Am besten klärt man all dies noch nicht in großer Runde, sondern vereinbart ein Treffen von etwa 30 Minuten Dauer mit einzelnen Personen. Die kurz bemessene Zeit sorgt dafür, dass sich auch kurz gefasst werden muss, wodurch eine automatische Priorisierung vorgenommen wird. Indem man nur mit einzelnen Personen spricht, kann man den Gesprächspartnern konzentriert zuhören und vermeidet eine Konkurrenzsituation, in der meist nur die kommunikationsstärksten Charaktere gewinnen. Gestaltet man die Termine dann eventuell als gemütliche Gespräche in einer Kaffeerunde, nimmt man ihnen außerdem den offiziellen Charakter und vermeidet somit allzu große Erwartungen und möglicherweise komplexe Prozesse, wie sie beim Austausch von Informationen zwischen Abteilungen in großen Unternehmen gelegentlich notwendig sind.

Ein einfaches „Ich wollte nur mal hören, wo der Schuh aus deiner Sicht drückt, bevor ich ein allzu großes Rad drehe“ ist viel unverfänglicher als ein stundenlanges Abstimmungsmeeting über Abteilungen hinweg. Letzteres kann immer noch geschehen, falls sich herausstellt, dass die Situation dies auch notwendig macht. Denn zu bedenken ist außerdem, dass man mit solchen Meetings sehr schnell eine Kette von Ereignissen lostritt. Dann wollen eventuell zusätzliche Personen die Richtung vorgeben oder die damit verbundenen Budgets für andere Dinge nutzen. Auch kann es sein, dass man in den Kurzgesprächen feststellt, dass das Vorhaben insgesamt zu groß ist, nur gibt es kein Zurück mehr, falls zu viele Personen beteiligt sind oder Dinge versprochen wurden.

Man merkt schon, bei der Sanierung von Software ist ein gewisses politisches Gefühl gefragt. Je größer das Unternehmen und die Software, desto größer auch die damit verbundenen Kommunikationsaufwände. Die Wahrheit ist aber, dass man sich diesem Aspekt lieber zu Beginn der Sanierung stellen sollte als am Ende. Denn je nach Umgebung kann ein einzelner Kritiker ausreichen, dass das Vorhaben beerdigt und im schlimmsten Fall den ursprünglichen Initiatoren als Makel des Fehlschlags angeheftet wird. Daher ist es sinnvoll, schon hier schrittweise und unverfänglich die Lage zu sondieren.

Kenne die Probleme

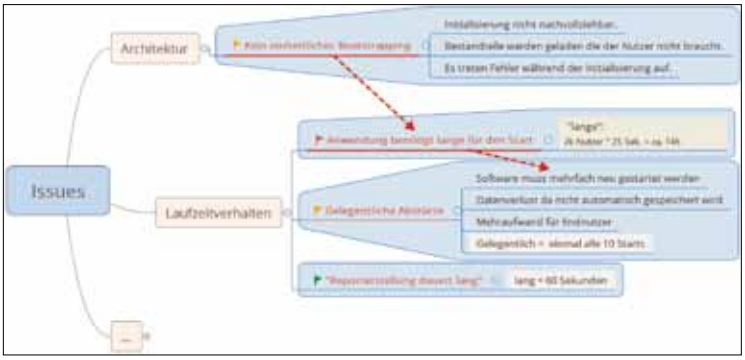
Die wichtigsten Gruppen und deren Motivation sowie Schmerzpunkte sind nun bekannt. Vermeintlich ist es also an der Zeit, die Schaffenskraft walten zu lassen und die Probleme zu beseitigen. Bevor dies aber geschehen kann, bedarf es noch einer belastbaren Aufstellung dieser Probleme und ihrer Auswirkungen.

Dazu eignet sich ein Issue Backlog oder eine Issue Map. Ersteres ist eine Liste, wie man sie mit Excel verwalten kann, Letzteres lässt sich mit Mind-Mapping-Tools sehr gut umsetzen. In der Praxis hat es sich sogar bewährt, zunächst alles in einer Liste zu sammeln, um diese anschließend mithilfe einer oder mehrerer Mindmaps zu strukturieren. In beiden Fällen sollte man jedoch von vornherein versuchen, die eigentliche Problembeschreibung von den damit verbundenen Auswirkungen zu trennen. Denn ein Problem kann sich für unterschiedliche Stakeholder unterschiedlich äußern.

Nehmen wir zum Beispiel eine geringe Testabdeckung. Dabei ist das Ausgangsproblem offensichtlich, dass es zu wenige automatisierte Tests gibt, wodurch die Entwickler aufwendige manuelle Tests durchführen müssen. Weil sie aufwendiger sind, werden sie selten durchgeführt und damit wiederum Fehler übersehen. In der Praxis betrifft dies aber nicht nur die Entwickler, sondern auch die Benutzer, welche in ihrer Arbeit behindert werden, wenn Fehler es bis in die Produktion schaffen. Möchten sie dies verhindern, müssen die Fachbereiche oder Auftraggeber selbst sehr umfangreiche Abnahmetests durchführen. An eine Umstrukturierung ist in so einer Situation kaum zu denken, da jede Änderung die Stabilität beeinträchtigen kann. Man sieht schon, ein Problem kann viele Auswirkungen haben, und teilweise sind Probleme selbst Auswirkungen anderer Probleme. Indem man nun aber die Auswirkungen den Zielgruppen zuordnet, kann man mögliche Gegenmaßnahmen auf Basis der Erlebnisse und Erkenntnisse der jeweiligen Zielgruppe rechtfertigen.

Bild 4 zeigt einen kleinen Ausschnitt einer möglichen Issue Map samt eines Worst-Case-Szenarios. Bei dieser Map wurden die Problemstellen in unterschiedliche Kategorien eingeteilt, wobei ursprünglich neben der Architektur und dem Laufzeitverhalten noch der Entwicklungsprozess, die Datenbank, Security und die Anbindung von Drittsystemen eine Rolle spielten. Der Umfang des beschriebenen Worst-Case-Szenarios ergab sich dann auch erst aus der Betrachtung aller Teilprobleme. So war den Entwicklern zwar klar, dass die Anwendung vergleichsweise lange für den Start benötigt, da ihnen aber nicht bewusst war, wie häufig die Anwendung gestartet wurde, ergab sich auch kein Problembewusstsein. Spätestens aber in Kombination mit den gelegentlichen ▶

Abstürzen und der hohen Anzahl an Nutzern gewann das Thema an Brisanz, da eine erhebliche Menge an Arbeitszeit täglich nur mit dem Start der Anwendung verloren ging. Diese Zusammenhänge wurden beim Erstellen der Karte über Querverweise kenntlich gemacht. Darüber hinaus wurde für die Priorisierung der Probleme ein Fähnchensymbol genutzt. Grüne Fähnchen bedeuten eine geringe Priorität, gelbe sehen einen Handlungsbedarf, der bei rot markierten Problemen akut ist. Weiterhin zeigen sich in den Problembeschreibungen weiche Formulierungen wie „lange“ oder „gelegentlich“. Möchte man diese Dinge richtig einschätzen können, muss man die subjektiven Beschreibungen mit Messungen unterlegen und prüfen, wie lange der Start denn durchschnittlich dauert und wie häufig die Abstürze tatsächlich auftreten. Selbst wenn diese Werte nicht eindeutig erfasst werden können, lohnt es sich dennoch, die weichen Formulierungen mit Schätzwerten zu unterlegen, wobei in jedem Fall transparent gemacht werden muss, wie gemessen beziehungsweise geschätzt wurde, um die Werte später noch einmal überprüfen zu können. Da es sich hierbei nicht um Auswirkungen handelt, wurden diese Informationen mit einem gelblichen Hintergrund in die Karte eingetragen. Sie könnten aber genauso gut direkt in der Problembeschreibung stehen.



Ausschnitt aus einer Issue Map (Bild 4)

Priorisiere

Nun, da eine tatsächliche Übersicht der bestehenden Probleme existiert, kann man beginnen, über mögliche Gegenmaßnahmen nachzudenken und diese ebenfalls einzutragen. Hinzu kommen noch deren mögliche Aufwände, wobei man sich hier noch nicht die Mühe zu machen braucht, in konkreten Zahlen zu argumentieren. T-Shirt-Größen von S bis XL oder ähnlich abstrakte Größenverhältnisse sollten zunächst ausreichen, da das Ziel der Schätzung nur darin besteht, die Priorisierung der einzelnen Maßnahmen zu unterstützen. Erst wenn sich anhand der Schweregrade der Auswirkungen akuter Handlungsbedarf herauskristallisiert, hat es Sinn, die damit verbundenen Maßnahmen auch mit groben Zeitschätzungen in Tagen, Wochen oder Monaten zu versehen. Hilfreich ist dabei immer, einen Bereich mit Von-bis-Zeiten zu verwenden, um die Erwartungshaltung etwas zu steuern, ohne übertriebene Puffer einbauen zu müssen. Mit diesen Informationen kann nun noch eine weitere Runde mit den Product Ownern, Fachbereichen et cetera gedreht werden, damit jene den Umfang und die Zusammenhänge der Ist-Situation verstehen. In diesem Fall können die Maßnahmen dann noch einmal anhand der langfristigen Ziele der Stakeholder ausgerichtet werden.

Des Öfteren stellt man beim Erstellen der Issue Map fest, dass sich viele Punkte ergeben, für die weitere Nachforschungen notwendig sind. Insbesondere die Definition des Sollzustandes macht mehr Arbeit, als es zunächst den Anschein hat. Dies fällt insbesondere im Bereich der Softwarearchitektur auf. Denn während man bei neuen Applikationen auf Basis eines sich ständig fortentwickelnden emergenten Designs arbeiten kann, muss bei Bestandssoftware darauf geachtet werden, welche Strukturen vorhanden sind und wie diese bereits miteinander verwoben sind. Oft hängen viele Bestandteile zusammen, und diese Querverbindungen müssen bei Anpassungen berücksichtigt werden. Es ist also sinnvoll, bei der Priorisierung auch darauf zu achten, inwieweit Arbeiten auch als Vorarbeiten für andere Sanierungsaufgaben dienen können. Auf diese Weise kann man sich von kleineren und leichter umzusetzenden Arbeiten schrittweise den größeren, komplexeren und damit zeitaufwendigeren annähern.

Vor einer Sache sei aber aus leidiger Erfahrung gewarnt: Man sollte vorsichtig sein bei der Sanierung auf Ebene des UI. Für viele Nutzer ist die Benutzeroberfläche der Anwendung eben jene Anwendung. Schnell wird davon ausgegangen, dass die Software sich in Summe besser verhält, nur weil die Oberfläche angepasst wurde, weil der Nutzer nicht einschätzen kann, dass es hinter dieser Oberfläche noch viel mehr Logik gibt, die noch immer schlecht strukturiert sein kann. Hinzu kommt, dass diese Oberfläche auch von jedem gesehen wird und somit alle Teilnehmer einen Beitrag zur Qualitätseinschätzung leisten können. Ausgelöst durch die Änderungsmaßnahmen werden aber die geänderten Bereiche stärker beachtet, wodurch auch Fehler gefunden werden, die eventuell schon länger existieren. Nun ist es für unbedarfte Betrachter naheliegend, dass solche Fehler erst durch den Umbau entstanden sind. Im ungünstigsten Fall kann dadurch der Eindruck entstehen, dass die Umbaumaßnahmen die Software instabiler und unordentlicher gemacht haben, obwohl die Stabilitätsprobleme schon länger existieren und erst die neuen Maßnahmen überhaupt die Transparenz geschaffen haben, auf Basis derer sie sichtbar werden.

Schulden über Schulden

Generell zeigt sich hier eines der größten Problemfelder der Sanierung von Software: die Transparenz. Sollte sie in einem Umfeld entstanden sein, in dem kein strukturierter Anforderungs-, Test- oder generell Entwicklungsprozess existiert hat, ist die Datenbasis, auf der Entscheidungen getroffen wurden, eher vage. Solche Umgebungen haben dann also nicht nur mit technischen, sondern auch mit Test-, Anforderungs- und Dokumentationsschulden zu kämpfen. Diese müssen zunächst beseitigt werden, und das bedeutet einen erheblichen Mehraufwand, der oft unterschätzt wird. Hinzu kommt, dass

man nicht so recht sicher sein kann, wie die vorhandenen Strukturen eigentlich gemeint waren und ob sie tatsächlich tun, was sie sollen. Gibt es dann auch noch Personen, die der Sanierung kritisch gegenüberstehen oder ist das Budget für die Sanierung zur knapp bemessen, ergibt sich daraus schnell eine Situation, in der man die Übersicht verliert und einfach nur noch auf die Veränderungen reagiert, statt zielgerichtet zu agieren. Nimmt man sich also zu viel vor und kontrolliert nicht regelmäßig die Erreichung der eigenen Ziele, läuft man Gefahr, in einen Zustand der Planlosigkeit abzurutschen, der im Worst Case in einer Panik endet, bei der jeder Beteiligte versucht, die eigenen Schäfchen ins Trockene zu bringen.

Dieser Umstand sollte um jeden Preis vermieden werden. Um das damit verbundene Risiko schon von Beginn an zu minimieren, sollten alle Entscheidungen in Protokollen schriftlich dokumentiert und zentral für alle Beteiligten einsehbar abgelegt werden, damit sie auch zu einem späteren Zeitpunkt nachvollzogen werden können. Absprachen müssen außerdem verbindlich sein und sollten von allen Beteiligten getragen werden. Sollte dies nicht möglich sein und sollte sich bereits bei den Anfangsgesprächen eine größere Abwehrhaltung gegenüber den Anpassungen zeigen, dann muss man die Ziele kleiner definieren und auch den beteiligten Personenkreis einschränken.

Hilfreich ist es hierbei, zunächst eine Art Beispiel zu schaffen, anhand dessen eine Sanierung geübt und mögliche Erfolge erläutert werden können. Diese Success Stories sind aber nicht nur wichtig, um weitere Unterstützer zu werben, sie sind auch für das beteiligte Team sehr wichtig.

Die Restrukturierung von Quellcode zählt zu den weniger beliebten Aufgaben in der Softwareentwicklung, weil sie einen sehr hohen Analyseaufwand mit zeitgleich sehr hoher Fehlerwahrscheinlichkeit bedeutet. Die Chance, dass man also etwas nicht versteht oder glaubt etwas zu verstehen und es dann doch Probleme macht, ist sehr hoch. Da sind Erfolge echter Balsam für die Seele. Es ist demnach auch wichtig, die Moral im Team zu stärken, um nicht den Eindruck entstehen zu lassen, die Teammitglieder befänden sich auf einer Art Abstellgleis. Dieses Gefühl kann bei einigen Entwicklern schnell aufkommen, wenn sie sich nicht mehr mit den neuesten Technologien, Mustern und Frameworks beschäftigen dürfen. Vor allem dann, wenn sich die Restrukturierung eben nicht durch die Ablösung von altem Code durch neuen, sondern mehr durch die Verwaltung von Missständen auszeichnet.

Let's do it

Das eigentliche Doing der Umstrukturierung soll an dieser Stelle nicht ausgiebig besprochen werden, dafür gibt es eine ganze Reihe unterschiedlicher Restrukturierungsmuster und Migrationsstrategien, die den Rahmen dieses Artikels sprengen würden. Rein organisatorisch ist es aber in jedem Fall wichtig, die Beteiligten auch bei einer Sanierung von Quellcode über den Fortschritt der Arbeiten zu informieren. Dies versteht sich eigentlich von selbst, geht aber dennoch in der Praxis zu oft unter. Eben weil sich Sanierungen nur selten an der Oberfläche der Anwendung zeigen und oft genug dazu führen, dass die Software am Ende eigentlich nur etwas besser läuft als zuvor, gibt es keine eindeutigen Indikatoren darauf, wie die Arbeit verläuft. Dabei lohnt es sich, die Messwerte aus der Issue Map zu verwenden und mit den neuen Werten zu vergleichen. Wurden dort konkrete Probleme angesprochen, sollten diese bestenfalls eben nicht mehr auftreten, und wenn doch, dann nur in einer verminderten Form. Wem man dann was wie erklären muss, ergibt sich somit ebenfalls aus der Issue Map. In diesem Zusammenhang ist eine direkte Vorstellung der Leistungsverbesserungen gegenüber den Beteiligten (inklusive Fachbereich) zu empfehlen oder zumindest eine Ergebnisdokumentation in Form einiger PowerPoint-Folien. Generell darf man die Themen Tests und Dokumentation auch in diesem Zusammenhang nicht vergessen. Es ist eine Sache, sich die Zustimmung zu einer Sanierung einzuholen und diese gegebenenfalls durch schlechte Codequalität und Ähnliches zu rechtfertigen. Es wäre dann aber fatal, die Arbeiten mit einer vergleichbaren Situation abzuschließen. Ebenfalls zur Dokumentation gehört eine Beschreibung der absehbaren Beeinträchtigungen, bevor die neuen Lösungen in den Produktivbetrieb übernommen werden. Dies gehört zwar eigentlich bereits in die Analysephase; da sich in Legacy Code immer auch weitere Fallstricke verbergen können, ergeben sich Planänderungen, die rechtzeitig kommuniziert und vor allem begründet werden müssen.

Zusammenfassung

Die Sanierung von Bestandssoftware kann durch den Austausch einzelner Codebestandteile oder Subsysteme geschehen. Sie hat viele unterschiedliche Ausprägungen und ist sehr stark von der Umgebung abhängig, in der sie geschieht. Dabei ist es wichtig, sich nicht nur auf rein technische, sondern auch auf fachliche Anforderungen zu konzentrieren. Um größere Umbauten innerhalb eines bestehenden Softwaresystems zum Erfolg zu führen, bedarf es einer umfangreichen Analysephase, bei der die tatsächlichen Anforderungen der neuen Lösung geklärt werden müssen. Einfache Bereinigungen im Kleinen gehören eigentlich nicht mit Personen außerhalb des Entwicklungsteams abgesprochen, schaffen sie doch nur unnötige Verhandlungsmasse über Themen, die nicht zur Verhandlung stehen sollten. Größere Änderungen müssen aber abgesprochen werden, um notwendige Anforderungen zu ermitteln und um die Projektbeteiligten frühzeitig auf die Auswirkungen der Anpassungen vorzubereiten. In diesen Fällen empfiehlt sich absolute Transparenz über Vorgehen, Ziele und erbrachte Leistungen. Dies schafft die notwendige Vertrauensbasis und beugt unnötigen Eskalationen vor. ■

Hendrik Lösch
ist Coach und Consultant bei Saxonia Systems sowie Fachautor. Er widmet sich bevorzugt Themen der Frontend-Entwicklung und Testautomatisierung. Sie erreichen ihn über Just-About.Net oder per E-Mail an **Hendrik.Loesch@saxsys.de**

dnpCode A1904Refaktorisieren