Heinrich C. Mayr, Martin Pinzger (Hrsg.): INFORMATIK 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 129

Software Qualität ist wie Schönheit

Sabine Wieland[1], Andreas Hartmann[2]

Abstract: Globalisierung, Digitalisierung, Industrie 4.0, Internet of Thinks, Big Data, Cyber Kriminalität, SOA, Cloud Computing und die Energiewende bestimmen den Alltag in Deutschland. Software Qualität wird von jedem Nutzer unterschiedlich empfunden und lässt sich genauso schlecht messen wie Schönheit. Die Gemeinsamkeiten von Software Qualität und Schönheit werden in diesem Artikel vorgestellt und daraus Schlussfolgerungen für die Verbesserung der Software Qualität gezogen. Der Artikel geht außerdem auf Methoden und Regeln zur Entwicklung von Software mit hoher Qualität ein.

Keywords: Software Qualität, Methoden und Regeln für Software Qualität, Software Qualitätsmerkmale, komplexe Software Systeme, Qualitätsmaße

1 Die Herausforderung der Digitalisierung

Mit Industrie 4.0 ist die nächste Ebene der Digitalisierung unserer Gesellschaft erreicht. Jeder Lebensbereich ist von der Digitalisierung betroffen und daher implizit abhängig von der Qualität der eingesetzten Software. Vom intelligenten Haus bis zum Internet der Dinge wird unsere Gesellschaft durch Software beeinflusst und der Mensch hinterlässt mit jeder Aktivität eine digitale Spur – er wird sprichwörtlich zum gläsernen Mensch.

Vielen Unternehmern sind die Sicherheitslücken ihrer IT Systeme nicht bewusst. Dennoch sind ebendiese Sicherheitslücken in den Software Systemen unterschiedlich häufig enthalten. Es genügt schon eine einzige veröffentlichte Schwachstelle und das gesamte System ist angreifbar. Große Datenmengen (Stichwort Big Data) werden erfasst und weltweit übertragen. Die Daten können manipuliert und missbraucht werden, wodurch ein hoher wirtschaftlicher Schaden entstehen kann. Es geht daher nicht nur darum, die Übertragung der Daten abzusichern, sondern auch die Daten verarbeitende Software vor Angriffen zu schützen. Dies gelingt am besten, wenn die Software selbst sicher ist.

Komplexe Softwaresysteme bedienen sich an Komponenten aus Klassenbibliotheken von Drittanbietern bzw. nutzen sie für das Design und die Entwicklung Services, Patterns und Frameworks. Softwaresysteme entstehen auch unter Verwendung von Open-Source-Komponenten. Diese Wiederverwendung hat viele Vorteile für die Entwicklung von komplexen Software Systemen. Bereits mehrfache genutzte und getestete Softwarebausteine können wiederverwendet und zu neuen Systemen

[1] Hochschule für Telekommunikation Leipzig, Gustav-Freytag-Str. 43-45, 04277 Leipzig, wieland@hftl.de
[2] Hochschule für Telekommunikation Leipzig, Gustav-Freytag-Str. 43-45, 04277 Leipzig, hartmann@hftl.de

kombiniert werden. Hier herrscht eine große Freizügigkeit, die durch verschiedene Lizenzmodelle unterstützt wird. Dabei beachten die Nutzer dieser Softwarekomponenten nicht die eingeschränkten Haftungsbestimmungen der Lizenzmodelle. Unter dem Gesichtspunkt der Haftung muss jede wiederverwendete Softwarekomponente im Zielsystem auf mögliche Fehler und Schwachstellen geprüft werden.

Die professionelle Cyberkriminalität nimmt stetig zu. Die Netzwerkangriffe sind um das Fünffache, von 7 Millionen im Mai auf 36 Millionen im August 2015 gestiegen [BSI14]. Dabei wird das Ziel, kritische Infrastrukturen zu treffen und deren Betrieb zu beeinträchtigen und zu stören, mit professionellen Methoden und hoher krimineller Energie verfolgt.

Im Jahr 2014 wurden laut HPI [Ho15] weltweit fast 6500 Sicherheitsschwachstellen in Software gemeldet – davon alleine 200 Sicherheitslücken für OpenSSL. Das ist positiv, denn jede gemeldete Sicherheitslücke kann und muss behoben werden. Ein zeitnahes Release mit der Fehlerbehebung bei OpenSSL zeigt den professionellen Umgang mit erkannten Sicherheitslücken. Software von Microsoft, Apple, Google, Mozilla, Oracle, HP, Adobe etc. reiht sich mit kritischen Sicherheitslücken in die Statistik [BSI14] ein. Unberücksichtigt bleiben Sicherheitslücken, die zwar bekannt sind, jedoch nicht gemeldet werden.

2 Die aktuelle Software Qualität

Entgegen diesem Hintergrund wird Software nur zu 30% getestet, uralte Software verwendet die nicht mehr gepflegt wird und Software ausgeliefert, bei der einfachste Entwicklungsregeln nicht berücksichtigt wurden.

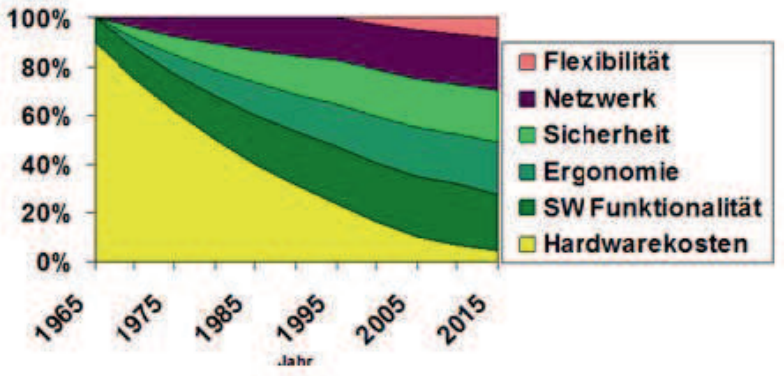

Abb. 1: Entwicklung der Software Komplexität

Allein diese Fakten zeigen, dass das Qualitätsbewusstsein für Software verbessert werden muss. Außerdem müssen geeignete Rahmenbedingungen für die Herstellung von

sicherer Software in allen Bereichen geschaffen werden.

2.1 Software Qualität ist Geschmackssache

Software Qualität wird sehr subjektiv bewertet. Jeder Nutzer stellt andere Anforderungen an die von ihm genutzte Software. Diese Anforderungen werden geprägt durch individuelle Erfahrungen, bereits genutzte Software und durch ein ganz persönliches Nutzerverhalten. Entsprechend der 80/20 Regel verwenden 80% der Nutzer 20% der Funktionalität, 20% der Nutzer ca. 80% der Funktionalität der Software. Nur sehr selten wird die volle Funktionalität von Software genutzt.

Abb. 2: Schönheit [Fo16, Wi16, Ac16]

Verschiedene Stakeholder – meist Gruppen von Interessenten – bestimmen die Qualitätsanforderungen, die an Software gestellt werden. Neben dem Nutzer werden der Auftraggeber und der Entwickler bei jedem Projekt mit berücksichtigt. Das spiegelt sich z.B. auch bei den Projektmanagementrollen im Vorgehen Scrum, dem Scrum Master, dem Product Owner und dem Entwicklerteam, wieder. Andere Stakeholder dagegen werden oft vergessen oder zu spät berücksichtigt. Dazu zählen der Systemadministrator, der Betriebsrat, der Datenschützer oder der Gesetzgeber. Eine rechtzeitige Integration aller Stakeholder in den Entwicklungsprozess ermöglicht die Entwicklung von sicherer und guter Software. Sobald einige Stakeholder nicht berücksichtigt werden, leidet das Immunsystem der Software darunter, vergleichbar mit Bild rechts in Abbildung 2. Ein schwaches Immunsystem ist bei Software an einem erhöhten Testaufwand, erhöhten Wartungsbedarf, vielen Sicherheitsschwachstellen und mangelnder Dokumentation zu erkennen.

Unser Empfinden sagt uns - nur ein gesundes Gesicht ist ein schönes Gesicht. Gesundheit muss jedoch auch gepflegt werden, denn ohne Pflege und gesunde Ernährung kann die Gesundheit nicht erhalten bleiben. Auch hier findet sich eine Parallele zur Software. Software *altert* über die veränderlichen Anforderungen (Changes) und durch Veränderungen in der Umgebung. Erfolgt keine Wartung und Anpassung an die veränderten Bedingungen *veraltet* die Software und wird schließlich unbrauchbar.

Software Qualität ist prozess- und produktbezogen. Eine MS Access Datenbank kann prima im privaten Gebrauch und auch noch für eine Handbibliothek mit 500 Büchern und einem Nutzer funktionieren. MS Access ist aber mit einer Anwendung für mehr als 100 Mitarbeiter, die gelegentlich Einträge vornehmen, überfordert. Hier bestimmt der Prozess, in dem die Software zum Einsatz kommt, die Anforderungen, die erfüllt werden müssen. Zum Beispiel steigen die Sicherheitsanforderungen, wenn personenbezogene Daten verarbeitet werden. Eine flexible Skalierung bei wechselnden Nutzerzahlen wird heute von fast jeder Software verlangt. Auch Echtzeitanforderungen variieren je nach Geschäftsprozess.

Datenbanksysteme, Editoren, Betriebssysteme, Entwicklungssysteme, Data Warehouse und Firmware wurden für ein bestimmtes Anwendungsgebiet entwickelt. Für jedes Anwendungsgebiet gibt es verschiedene Softwareprodukte - bei den Datenbanken z.B. Informix, Oracle, MySQL, Access etc. Jedes dieser Produkte weist Besonderheiten auf, die es einmalig machen. Als Beispiel seien zwei Editoren genannt, die sehr verschieden sind und jeder für sich Vor- und Nachteile ausweist. Der zu Unix gehörende vi ist schlank, robust und sehr leistungsfähig; besitzt aber nur eine rudimentäre Bedienerführung. MS Word dagegen enthält viele verschiedene Funktionen und eine ansprechende Oberfläche, ist jedoch fehleranfällig und vergleichsweise weniger robust.

Da Software immateriell ist, kann sie (theoretisch) bei Benutzung nicht verschleißen. Ihr fehlen wie die physikalischen Grenzen auch mechanische Teile, die durch Reibung verschleißen können. Software verschleißt nur durch Programmierfehler. Zum Beispiel wenn Speicherbereiche nach Nutzung nicht gelöscht und freigegeben werden, wenn ungewollte Wechselbeziehungen zwischen Programmteilen existieren.

Passt eine Software gut in den zu unterstützenden Geschäftsprozess entstehen neue Anforderungen beim Anwender, da angrenzende Geschäftsprozesse ebenfalls bearbeitet werden, oder noch eine Statistik fehlt. Auch während dem Entwicklungsprozess verändert sich die Umwelt, in der die Software zum Einsatz kommt. Ein Betriebssystem Update wird eingespielt, eine Markterweiterung steht an und damit wird eine weitere Sprache benötigt etc. Jeden Monat ändern sich auf diese Art 5% der Anforderungen. Das macht die Software Entwicklung zu einem komplexen, zeitkritischen Prozess.

2.2 Qualitätsmerkmale nach ISO Standard

Software Qualität ist zunächst in der ISO/IEC Norm 9126 [ISO11] mit 6 Qualitätsmerkmalen definiert, siehe Abbildung 3, wobei diese Norm in den Standard ISO/IEC 25000 überführt wurde. Die Definition berücksichtigt auch, dass Software immateriell ist. Software wird wie ein Kunstwerk entwickelt. Jede Software wird als Einzelstück entwickelt. Hier gibt es viele Parallelen zur Baubranche. Jedes Gebäude ist auf Grund seiner Lage, Funktion und Ausführung einzigartig. Genormte Teile vereinfachen die Konstruktion der Gebäude. Auch für das Software Engineering existieren Standards und „genormte Teile“, wie Schnittstellendefinitionen, Klassenbibliotheken etc. Das bedeutet, dass alle Software Services / Module / Klassen /

Methoden etc. nur mittels Nachrichten und Funktionsparameter kommunizieren. Sobald alle Variablen eines Systems private sind und nur von den klasseneigenen Methoden verändert werden können, steigt nicht nur die Wiederverwendbarkeit dieser Services, sondern auch deren Software Qualität und Sicherheit!

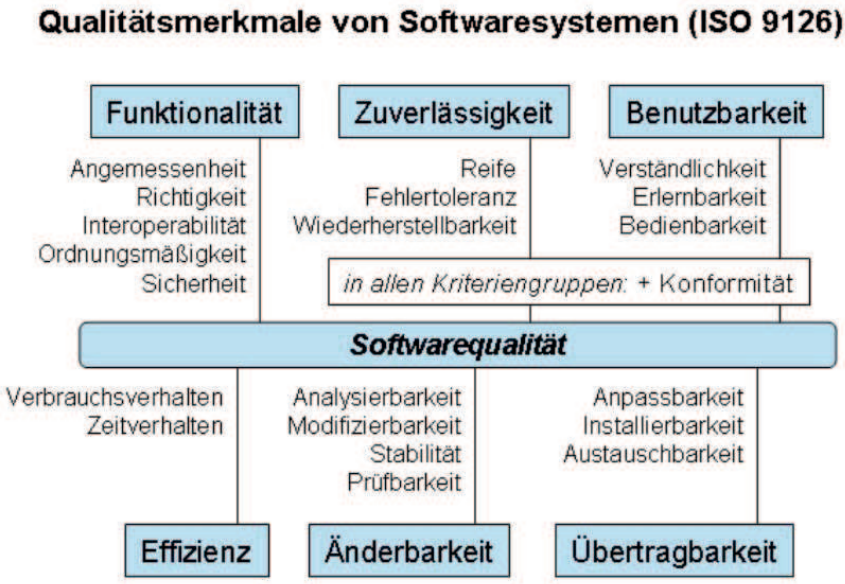

Abb. 3: ISO/IEC Standard 9126 [ISO11]

Es wird zwischen inneren und äußeren Qualitätsmerkmale unterschieden. Die äußeren Qualitätsmerkmale entsprechen den im ISO/IEC Standard 9126 beschrieben Merkmalen. Sie werden auch vom Nutzer wahrgenommen und bestimmen die Nutzerakzeptanz. Die inneren Qualitätsmerkmale beschreiben die innere Struktur der Software und definieren die Fehleranfälligkeit, Robustheit und Änderbarkeit der Software. Zum Beispiel können objektorientierte Strukturen mit den Metriken nach Chidamber und Kremerer [CK91] gemessen werden. Mit den Metriken WMC (Weighted Methods per Class), RFC (Response for a Class), DIT (Depth of Inheritance Tree), NOC (Number of Children), etc. kann die Komplexität und damit die Fehleranfälligkeit eines Software Systems bestimmt werden. Jede Metrik muss dabei folgende Kriterien [Ba08] erfüllen:

- Objektivität, keine subjektiven Einflüsse sind möglich,
- Zuverlässigkeit, liefert bei Wiederholung das gleiche Ergebnis,
- Validität, lässt eindeutige Rückschlüsse auf die gesuchte Kenngröße zu,
- Normierung, nutzt eine Vergleichsskala,

- Vergleichbarkeit, kann mit anderen Maßen in Relation gesetzt werden,
- Ökonomie, die Messung ist mit geringem Aufwand möglich,
- Nützlichkeit, die Aussage der Messung erfüllt praktische Bedürfnisse.

Die im Standard beschriebenen Teilmerkmale können durch Qualitätsindikatoren bzw. Metriken mess- und bewertbar gemacht werden, wobei quantifizierbare Qualitätsindikatoren mit Hilfe von Qualitätsmaßen gemessen werden. Die im Standard beschrieben Qualitätsmerkmale werden vom Kunden bzw. Nutzer in den NFA (Nichtfunktionalen Anforderungen) beschrieben. Die Definition von NFA fällt dem Kunden noch schwerer als die Bestimmung von funktionalen Anforderungen (FA). Der Standard ISO/IEC 9126 gibt mit seiner klaren Definition der Qualitätsmerkmale wichtige Hinweise für die Definition von NFA und FA. Checklisten, Schablonen und Metriken helfen bei der Definition der Anforderungen. Nur klar definierte Anforderungen bilden die Grundlage für eine nützliche Qualitätsmessung.

2.3 Spagat zwischen Wissenschaft und Kunst

Soweit die Theorie zur Entwicklung von guter Software. Heutige Software Systeme sind mit Hundertwasserhäusern vergleichbar. Es wird absichtlich von den Standards abgewichen, um die Genialität des Entwicklers darzustellen. Besonders Apps – Software für mobile Geräte – werden von Quereinsteigern entwickelt, die nur wenige Methoden und Regeln der Software Entwicklung kennen und keine entsprechende Ausbildung haben. Ein Beispiel dafür ist das Crowdtesting, bei dem der Betatest durch eine Vielzahl von Nutzern durchgeführt wird. Diese Tests erfolgen meist ohne Testplanung, definierten Testfällen und ausreichender Dokumentation. Das Crowdtesting hat den Vorteil, dass eine Vielzahl von verschiedenen Kombinationen aus Hardware, Betriebssystem, Versionen, Providern etc. getestet werden können. Die Testvielfalt entspricht damit auch den in den Zielgruppen genutzten Konfigurationen. Ein Crowdtest ist besser als gar kein Test.

Man nehme ein wirklich flexibles, frei orchestrierbares SOA-System, bei dem je nach Bedarf Services ausgetauscht werden. Wie soll die Funktionalität dieses Systems getestet werden, wenn zwischen den einzelnen Services nicht dokumentierte Abhängigkeiten existieren? Die Variantenvielfalt steigt ins unendliche und dann werden tatsächlich nur 30% des Systems getestet. Software Entwickler sind sehr intelligente, kreative Designer von neuen Systemen, die neben der Funktionalität auch immer das Ego des Entwicklers erfüllen sollen. Künstler – also auch Software Entwickler – halten sich nicht gern an Gebote und Regeln. Diese schränken die Möglichkeiten und Kreativität ein. Daraus entsteht der Spagat zwischen der Anwendung wissenschaftlicher Erkenntnisse und der kreativen Entwicklung von neuem.

Abb. 4: Hundertwasserhaus in Magdeburg [Wi16b]

Der Widerspruch ist auf die besondere Eigenschaft der Software zurückzuführen – Immaterialität. Jede beliebige Software – auch sogenannte Standardsoftware wird nur einmal entwickelt. Danach kann sie prinzipiell ohne Veränderung unendlich oft kopiert werden. Das bedeutet im Umkehrschluss – Software ist immer einmalig und jede Software wird neu durch einen kreativen Prozess entwickelt. Wenn tatsächlich heute jede Software von Grund auf neu entwickelt wird, dann sind Technologien wie SOA und Cloud Computing nicht realisierbar. Denn SOA als auch Cloud Computing basieren auf lose gekoppelten Software Services, die nur mittels Nachrichten kommunizieren.

Wie jede kreative Tätigkeit benötigt auch ein Software Entwicklungsprojekt eine Pause und die Besinnung auf das große Ganze. Das Whiscy-Syndrom steht im Zusammenhang mit dem Zeitfaktor bei der Entwicklung von Software-Projekten. Nach dem Motto „Zeit ist Geld“ wird versucht, die Mitarbeiter eines Software-Projektes zum effektiven Arbeiten anzuhalten. Dabei ist die Frage *Was machen die Mitarbeiter eigentlich den ganzen Tag?* sicher berechtigt. Dass es nicht ausreicht, Befehle aneinander zu reihen, sondern dass diese Befehle auch die Aufgabe lösen müssen, kann dabei schnell in Vergessenheit geraten.

Schnell setzt sich beim Management der Irrglaube fest, die Entwickler sind nur produktiv, wenn sie Befehle schreiben, also *codieren*. Damit werden die Erfahrungen von vielen Projekten missachtet die besagen, dass die Implementierung den geringsten Zeitaufwand bei der Software-Entwicklung erfordert, siehe Abb. 5. Einen Befehl zu schreiben dauert keine Minute, den richtigen Befehl zu finden kann insgesamt (mit Anforderungsanalyse und Test) über eine Stunde dauern. Der Name Whiscy-Syndrom wurde von der Frage *Why isn`t Sam coding yet*? abgeleitet und macht mit einem Augenzwinkern auf das Problem aufmerksam. Dass jeder kreativen Tätigkeit eine Phase des Überlegens, Nachdenkens, aber auch der Recherche und Untersuchung vorausgehen

muss, sollte nicht vergessen werden.

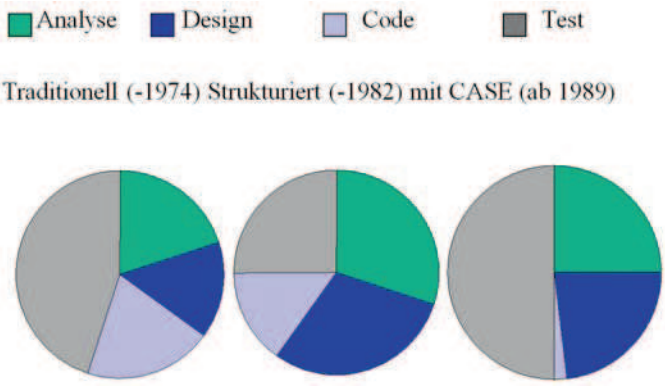

Abb. 5: Zeitaufwand für die Phasen Analyse, Design, Code und Test im SWE Prozess

Dass einfache Gemüter diesen Teil der kreativen Tätigkeit gern vernachlässigen, ist nicht neu. Auch Leonardo da Vinci hatte mit seinen Auftraggebern so seine liebe Not, die ihn zu dem Ausspruch zwang, „dass erhabene Geister bisweilen am meisten schaffen, wenn sie am wenigsten arbeiten“. Nachdenken kann schnell als „In die Luft starren“ – also Nichtstun gedeutet werden, da es nicht durch eine entsprechende Mimik deutlich wird, ganz im Gegenteil. Viele kreative Köpfe machen in den schöpferischsten Phasen ein ausgesprochen dummes Gesicht.

In diesem Ausspruch von Leonardo da Vinci steckt auch, das für eine hervorragende kreative Leistung auch Phasen der Entspannung notwendig sind. Ein Entwickler, der ständig unter Zeitdruck steht, wird in seiner Kreativität und Produktivität schnell nachlassen. Dieses Phänomen wird auch in den sogenannten Startup- Unternehmen beobachtet. Nach mehreren Monaten, manchmal auch Jahren intensivster Arbeit ohne Pause und mit reichlichen Überstunden sind auch die kreativsten Köpfe ausgepowert und brauchen Erholung.

Der nächste Aspekt, der mit zum Whiscy-Syndrom gehört, ist die Konzentration auf ein Fachgebiet. Der Blick über den Tellerrand kann zu neuen Erkenntnissen führen. Ein einseitig nur mit Rechnern und dessen Problemen Beschäftigter nutzt keine Erkenntnisse aus anderen Fachgebieten. In der Informatik hat diese Isolation der Fachkräfte und Experten schon verheerende Ausmaße angenommen. Hier kennt der Datenbank-Spezialist die neuesten Erkenntnisse auf dem Gebiet der Betriebssysteme nicht und der Netzwerk-Spezialist hat keine Ahnung, wie ein Datenbank-Management funktioniert. Erst recht werden Erkenntnisse der Randgebiete, wie die Künstliche Intelligenz, übersehen. In der Nutzung möglicher Synergien liegt ein sehr großes Potential, das beachtet werden sollte.

3 Methoden und Regeln für Software Qualität

Das Teufelsquadrat beschreibt die Wechselwirkungen zwischen den Größen Qualität, Quantität, Projektdauer, Projektkosten und Produktivität. Die Größen Qualität, Quantität, Kosten und Zeit stehen in sehr enger Beziehung zueinander. Eine Veränderung einer Größe wirkt sich entsprechend auf die anderen aus.

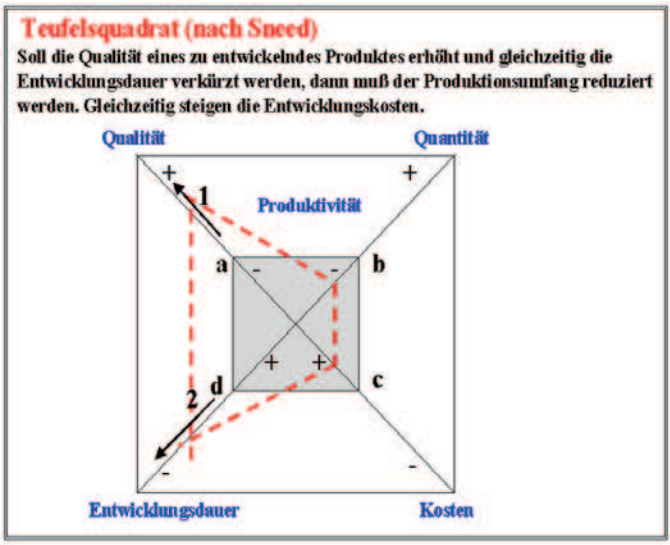

Abb. 6: Teufelsquadrat nach Sneed [Ba09]

Hohe Qualitätsanforderungen können nur mit einem ausreichenden Budget und ausreichender Zeit realisiert werden. Die Quantität wird durch die funktionalen Anforderungen definiert und stellt den Umfang des Software Systems dar. Für die meisten Software-Projekte sind sowohl die Qualitätsanforderungen als auch der Lieferzeitpunkt vom Auftraggeber vorgegeben. Eine entsprechende Kalkulation des Preises für das Softwarepaket (Funktionsumfang und geforderte Qualität) und die anschließende Diskussion mit dem Auftraggeber ist die erste Hürde, die das Projektmanagement bewältigen muss. Ein Missverhältnis zwischen den Anforderungen und den für die Entwicklung zur Verfügung stehenden Zeitraum kann das Softwareprojekt zum Scheitern verurteilen. In einem Softwareprojekt ist es nicht möglich, mit doppelt so vielen Entwicklern die Entwicklungszeit zu halbieren, da der höhere Kommunikationsbedarf zwischen den Entwicklern zu einer Verzögerung führt. „Ein verspätetes Softwareprojekt wird durch die Hinzunahme eines zusätzlichen Entwicklers weiter verzögert." [Ba09] Dieses Phänomen wurde in vielen Softwareprojekten beobachtet und vielfach beschrieben. Der neue Entwickler muss mit dem Projekt vertraut gemacht werden, ehe er bei der Entwicklung mithelfen kann. In einer späten Entwicklungsphase ist das mit hohem Aufwand verbunden. Auf den Fortschritt eines

Softwareprojektes wirken sich neue Qualitätsanforderungen und Budgetkürzungen negativ aus.

3.1 Miss es oder vergiss!

Jede Anforderung, die in das Pflichtenheft aufgenommen wird, muss nachweisbar sein! Es muss eine klare Beschreibung vorliegen, wie diese Anforderung nachgewiesen / getestet werden kann. Ist die Anforderung nicht nachweisbar, soll sie nicht in das Pflichtenheft aufgenommen werden. Denn kann die Anforderung nicht nachgewiesen werden, braucht sie auch nicht realisiert werden. **Miss es oder vergiss es!** Um das Formulieren von Anforderungen zu vereinfachen, werden Anforderungsschablonen und UML-Diagramme oder die BPMN Notation verwendet. Je klarer die Anforderung formuliert ist, desto einfacher ist deren Nachweis in einem Test. „Die Anwendung muss performant sein“ oder „Die Anforderung muss beim Datenbankzugriff eine Antwortzeit unter 1 s erfüllen.“ Damit beginnt die Testphase bereits zur Zeit der Analyse, siehe Abb. 7. Während der Testplanung werden für jede Anforderung (Anforderungsgruppe) die geeigneten Testverfahren / Testfälle definiert und beschrieben. Auf dieser Grundlage werden in der Design Phase die Testumgebungen entwickelt und die Testdaten erstellt. Zur Implementierung folgt der Modultest. Vor dem Betrieb der Software erfolgt der Systemtest und während der Wartungsphase erfolgen die Wiederholungstests.

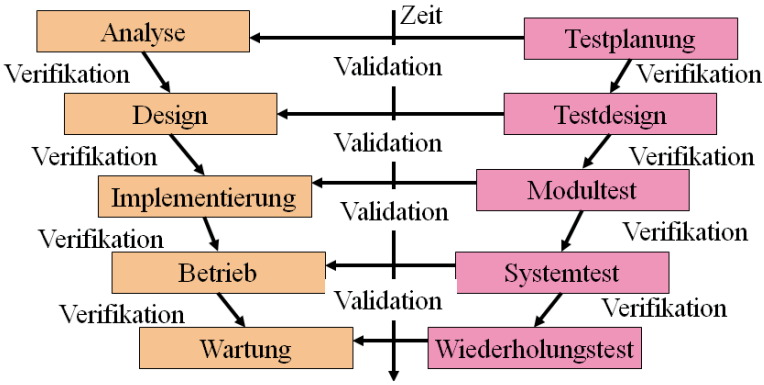

Abb. 7: angepasstes V-Modell

Codezeilen, die nicht getestet wurden sind „easter eggs“ und müssen gelöscht werden, denn diese nicht getesteten Codezeilen stellen Sicherheitslücken dar. Programmelemente die beim Testen der geforderten Funktionen und NFA nicht aufgerufen werden (dead code) müssen gelöscht werden, da sie ein erhöhtes Sicherheitsrisiko darstellen.

3.2 Technische Schulden

Ausprägungsart	Beschreibung	gesund / ungesund
Quelltextschulden	hohe Quelltextkomplexität, Nicht-Einhaltung des festgelegten Programmierstils	ungesund
Architekturschulden	Modularisierbarer Quelltext	ungesund
Dokumentationsschulden	Unverständliche Kommentare, defizitäre Informationsstruktur	ungesund
Testschulden	Nicht getesteter Quelltext (manuelle und Regressionstests)	gesund / ungesund
Schulden durch eine technologische Lücke	Nutzung veralteter Technologien	gesund / ungesund
Sicherheitsschulden	Nicht-Einhaltung der Sicherheitsrichtlinien	ungesund

Abb. 8: technische Schulden [Ba15]

Sobald der Blick primär auf den Liefertermin gerichtet ist und im Projekt notwendige Arbeiten aus Zeitmangel nicht realisiert werden können entstehen sogenannte „technische Schulden“ [KNO12], siehe Abb. 8. Wie bei der Kreditaufnahme müssen auch auf die technischen Schulden Zinsen gezahlt werden. Ein „Point of no return“ wird erreicht, wenn die zu erbringenden Zinsen höher als der aufgenommene Kredit sind. Durch die Dynamik im Software Entwicklungsprozess wachsen die Zinsen von technischen Schulden rasant. Die technischen Schulden müssen ausgeglichen werden, also die nicht realisierten Aufgaben müssen später nachgeholt werden. Erfolgt kein Ausgleich der technischen Schulden, wird die Wartung am System erschwert und die Qualität verschlechtert sich zunehmend (vgl. *Alterung*).

3.3 Das Geheimnisprinzip

Das Geheimnisprinzip fordert die Kapselung der Daten im Modul. Das ist u.a. einfach durch die Definition von privaten Attributen zu realisieren. Die Kopplung zwischen Modulen wird durch die Schnittstellen definiert. Je einfacher die Schnittstelle (Übergabe einfacher Datentypen per Nachricht) desto loser ist die Kopplung zwischen den Modulen. Solange einfache Datentypen (**Datenkopplung**) übergeben werden, ist das Software System gesund. Sobald Datenstrukturen übergeben werden (**Datenstrukturkopplung)** muss die Schnittstelle überprüft werden. Werden wirklich alle Datenelemente der Struktur vom Modul verwendet? Die Datenstruktur muss bei einer Weiterentwicklung oder Verwendung in einem anderen System simuliert werden das schränkt die Wiederverwendbarkeit ein.

Kopplung		wart-bar	wieder-ver-wendbar	Bindung	
Daten	lose	++	++	funktional	fest
Datenstruktur	lose	+	+	funktional oder sequentiell	fest
Kontroll	fest	-	-	kommunizierend oder problembezogen	lose
Hybrid	fest	--	--	problembezogen oder programmstrukturell	lose
global	fest	--	--	zeitlich	lose
Inhalt	fest	--	--	zufällig	lose

Abb. 9: Zusammenhang zwischen Kopplung und Bindung [My74]

Eine starke Kopplung verursacht eine lose Bindung / sehr geringen Zusammenhalt der Module [My74]. Eine starke Kopplung und lose Bindung in den Modulen eines Softwaresystems beruht meist auf Missachtung des Geheimnisprinzips und verursacht eine gegenseitige Beeinflussung der Module. Nach einigen wenigen Erweiterungen und Veränderungen durch Wartung treten nicht nachvollziehbare, kritische Runtime Fehler auf, die nur mit sehr hohem Aufwand gefunden und beseitigt werden können. Das Softwaresystem ist nicht mehr wartungsfähig. Auch die einzelnen Module / Services des Softwaresystems sind für eine Wiederverwendung in einem neuen Software System nicht geeignet, da sonst der Virus der gegenseitigen Beeinflussung in das neue System übertragen wird.

Das Geheimnisprinzip ist der Schlüssel für die Modularisierung da es Voraussetzung für die Abgeschlossenheit der Module und deren gegenseitigen Nichtbeeinflussung ist. Mit einem einfachen Software Qualitätsmesser können die öffentlichen Variablen in einem Softwaresystem ermittelt und mit der Gesamtanzahl der Variablen / Attribute verglichen werden. Viele Open-Source-Projekte weisen hier gravierende Sicherheitslücken auf, denn jedes öffentliche Attribut kann für einen Hackerangriff genutzt werden.

4 Fazit

Das tolle an Software ist: sie verschleißt nicht bei Benutzung – sofern sie frei von technischen Schulden ist! Hier ist sie besonders robust und widerstandsfähig! Bei Software gibt es (per Definition) keine Verschleißteile, die regelmäßig kontrolliert (jährliche Durchsicht) und ausgetauscht werden müssen! Software altert jedoch durch die Veränderungen in der Umwelt, durch die neuen Wünsche der Anwender, durch neue Gesetze und neue Marktsituationen.

Zusammenfassend können wir feststellen: Software Entwicklung mit hohem Qualitätsanspruch ist schwierig – um nicht zu sagen unmöglich!

1. Der Kunde weiß nicht was er will und kann es auch nicht klar ausdrücken.
2. Es gibt etwas über 20 Stakeholder, die die Qualität der Software definieren.
3. Die Anforderungen an die Software ändern sich (5% pro Monat).
4. Software ist immateriell und hat keine physikalischen Grenzen.
5. Projektfortschritt kann nur mittelbar über die Dokumentation gemessen werden.
6. Software Qualität ist prozess- und produktbezogen.
7. Software steht in enger Wechselbeziehung zu ihrer Umgebung (Prozesse, Hard- und Software).
8. Softwareprojekte sind einmalig – jede Software wird nur einmal entwickelt.
9. Software Entwicklung ist Kunst – ein kreativer Prozess.
10. Das Management unterschätzt den kreativen Prozess und unterliegt dem Whiscy-Syndrom.
11. Das Team ist nicht optimal zusammengestellt – es fehlen notwendige Skills
12. Das Geheimnisprinzip wird seit Jahren missachtet!

Wir erschaffen ein Monster! Ein weltumspannendes, das gesamte Leben und die Gesellschaft beherrschendes, von jedem manipulierbares Monster. Dieses Monster können wir auch nicht einfach abschalten, denn dann geht überall das Licht aus… Es gibt nur eine Lösung: Software unter Einhaltung der Methoden und Regeln des Software Engineering entwickeln und schrittweise alle andere Software ablösen. Hierzu sind SOA und Cloud Computing der richtige Weg, denn diese Technologien funktionieren nur mit lose gekoppelten Services!

Literaturverzeichnis

[Ac16] Acne Vulgaris, Cysic acne on the face, wikipedia, http://history.amedd.army.mil/booksdocs/wwii/internalmedicinevolIII/chapter20figure85.jpg, Stand: 25.6.2016

[Ba15] Bauer, M.; Ermittlung von technischen Schulden unter Verwendung von Metriken in einem agilen Software-Entwicklungsprojekt eines Provisionsabrechnungssystems , Bachelorarbeit, Hochschule für Telekommunikation Leipzig, 2015

[Ba08] Balzert, H.; Lehrbuch der Software Technik – Softwaremanagement, Spektrum Akademischer Verlag; Auflage: 2. Aufl. 2009

[Ba09] Balzert, H.; Lehrbuch der Software Technik – Basiskonzepte und Require-ments Engineering, Spektrum Akademischer Verlag; Auflage: 3. Aufl. 2009

[Bo11] Boehm, B.: in Sophist White Paper 2011

[BSI14] BSI Lagebericht 2014: https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf?__blob=publicationFile

[CK91] CHIDAMBER, Shyam R. ; KEMERER, Chris F.: *A metrics suite for object oriented design*. Rev. July 1993. Cambridge, Mass. : M.I.T. Center for Information Systems Research, 1993 (Sloan WP 3524-93)

[EM03] Andreas Essigkrug, Thomas Mey: Rational Unified Process kompakt. Spektrum Akademischer Verlag, Heidelberg u. a. 2003, ISBN 3-8274-1440-7.

[Fo16] iStockfoto; http://www.fitforfun.de/beauty-wellness/haut-haare/huebsch-schoenheit-ist-messbar-_aid_8673.html;

[Ho15] Hofmann, K.: IT Business, 18.3.2015: CeBIT 2015: Anwendungen mit den meisten Sicherheitslücken

[ISO11] ISO-Store ISO/IEC 9126-1:2001[1] This standard has been revised by: ISO/IEC 25010:2011

[KNO12] Philippe Kruchten, Robert L. Nord und Ipek Ozkaya. Technical Debt: From Metaphor to Theory and Practice. IEEE Computer Society, 2012.

[My74] Glenford J. Myers: Reliable Software through Composite Design. Mason and Lipscomb Publishers, New York 1974.

[Wi16a] Wikipedia, https://de.wikipedia.org/wiki/Alter, Stand 25.6.2016

[Wi16b] Wikipedia, https://de.wikipedia.org/wiki/Friedensreich_Hundertwasser#/media/File:Magdeburg_Hundertwasserhaus.jpg, Stand 25.6.2016