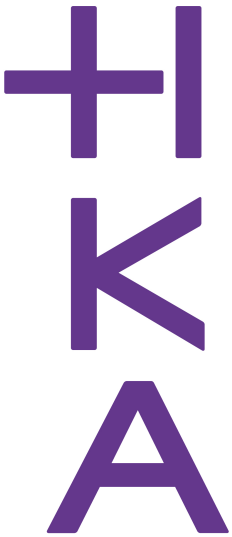
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

Studiengang
Medien- und Kommunikationsinformatik
und
Informatik



Aristocracy, Democracy and System Design

Jasin Aferkou - 71963 - afja1011@h-ka.de
Egzon Islami - 65124 - iseg1011@h-ka.de

Qualitätssicherung

14. November 2022

Inhaltsverzeichnis

1 Einleitung

1.1 Kathedralen

Der Großteil europäischer Kathedralen zeigt Zeichen verschiedener architektonischer Stile und Epochen. Dies liegt genau daran, dass diese gigantischen Bauprojekte über mehrere Jahrhunderte und damit über mehrere Generationen von Architekten und Erbauern durchgeführt wurden. Hierdurch fließen typischerweise die verschiedenen Geschmäcker und die Mode der verschiedenen Generationen ein. Das führt dann natürlich zu stilistischen Brüchen in der Gesamtstruktur dieser Gebäude. Die Kathedrale von Reims[1] sticht hierbei dadurch heraus, dass diese ein äußerst stimmiges Bild ausstrahlt. Diese `architektonische Integrität` konnte nur zustande kommen, indem die nachfolgenden Architekten und Erbauer darauf verzichteten ihre eigenen Ideen einfließen zu lassen, um genau die ursprüngliche Vision der ersten Architekten durchzusetzen. Sie wollten ein pures Design und damit ein stimmiges Gesamtbild erreichen. [1]

1.2 Software-Analogie

Aber was haben Kathedralen mit der Entwicklung von Softwareprojekten zu tun? Typischerweise bestehen die Teams, die insbesondere große Softwareprojekte umsetzen aus vielen Entwicklern. Softwareingenieure neigen hierbei auch dazu eine feste Meinung zu haben, wie bestimmte Probleme am besten zu lösen seien. Das ist natürlich menschlich, kann aber dazu führen, dass die Vision des Projektes aus dem Gleichgewicht gerät. Es entsteht eine konzeptionelle Uneinigkeit bei diesem Projekt. Hierbei liegt es allerdings nicht daran, dass das Projekt an einem bestimmten Punkt an nachfolgende Hauptdesigner oder Architekten überreicht wird, sondern an der Einteilung des Designs in viele Aufgaben, die von vielen verschiedenen Menschen bearbeitet werden. [1]

[1] Auch bekannt als Kathedrale de Reims

2 Konzeptionelle Integrität

2.1 Definition Integrität

Der Begriff konzeptionelle Integrität ist in keiner Literatur definiert. Selbst in keiner ISO-Norm zur Softwarequalität lässt sich der Begriff finden. Was sich aber finden lässt, ist eine Definition des Begriffs Integrität. Die mehrfach belegte Bedeutung macht es zu einem Homonym und ist dementsprechend nicht immer auf Anhieb passend zu verstehen, da sie von Themenfeld zu Themenfeld anders ausgelegt werden kann. Für unser Verständnis müssen wir den Begriff der Integrität aus der lateinischen Sprache übersetzen, die sich in unversehrt, intakt oder vollständig deuten lässt. Die Bedeutung dieser Vollständigkeit brauchen wir in unserem Kontext.

2.2 Integer vs. Float

Wir versuchen anhand dieser beiden Datentypen den Begriff Integrität noch einmal klarer darzustellen. Der Datentyp speichert nur ganze Zahlen, ohne Zwischenwerte. Er bildet also einen bestimmten Zahlenbereich innerhalb der ganzen Zahlen vollständig ab und bietet damit eine gewisse Abgeschlossenheit. Ein gutes Gegenbeispiel ist der Datentyp Float. Dieser bildet reelle Zahlen ab. Dieser Datentyp lässt also im Gegensatz zu ganzen Zahlen auch gebrochene Zahlen zu. Dadurch können auch ungenaue Werte entstehen. Beispielsweise deckt ein Integer mit 32 Bit alle ganze Zahlen zwischen -2.147.483.648 und 2.147.483.647 ab [3], während eine 32-Bit Fließkommazahl diese Bits in Exponent und Mantisse aufteilt, um dynamisch einen möglichst großen Bereich an Zahlen abzudecken. Dieser Datentyp kann hierbei wesentlich größere, oder kleinere Zahlen darstellen als ein 32-Bit Integer, weist aber genau dann, oder bei Zahlen, mit besonders vielen Nachkommastellen, Lücken auf. [2] Da reelle Zahlen zu dem `dicht liegen`, was bedeutet, dass zwischen allen zwei reellen Zahlen immer eine weitere reelle Zahl gefunden werden kann, ist es sonst auch gar nicht möglich einen Datentyp zu definieren, der einen beliebigen Zahlenbereich in den reellen Zahlen unter Verwendung einer fixen Speichergröße abdeckt.

2.3 Was versteht man unter konzeptioneller Integrität?

Was ist also unter konzeptioneller Integrität zu Verstehen? Den Grad von Kohäsion und Widerspruchsfreiheit von Anforderungen, die ein System in sich vereinen muss. Kohäsion bedeutet hierbei, wie eng die Anforderungen der Software zueinander in Beziehung stehen.

2.4 Beispiele für konzeptionelle Integrität

Folgende Beispiele sollen die konzeptionelle Integrität für das Verständnis greifbarer gestalten.

2.4.1 Kundendaten

Angenommen, es gibt eine API-Schnittstelle, die das Geburtsdatum eines Kunden in drei Parametern übergeben haben möchte. Auf dem Kontaktformular der Registrierung gibt es aber nur ein Feld, der auch noch kein Format für das Datum an sich vorschreibt. Dies ist ein offensichtlicher Widerspruch zwischen den zwei Schnittstellen, der für den Entwickler Aufwand bedeutet, um die Formate richtig zu konvertieren. Hier ist die konzeptionelle Integrität offensichtlich nicht gegeben und verursacht dadurch Mehrkosten.

2.4.2 Bestellsystem

Ein externes System soll nur fünfstellige Postleitzahlen akzeptieren. Dabei ist eine Anforderung, dass Bestellungen aus der Schweiz möglich sein sollten. Nun liegt das Problem in der Vierstelligkeit der Postleitzahl, die in der Schweiz nun mal üblich ist. Die Erschwernis folgt jetzt aus dem Code des externen Systems und der Anforderungsdokumentation, die einander nicht widersprechen sollten, da sonst die Verständlichkeit des Systems leidet.

2.4.3 Einheitlichkeit

Ein Beispiel, den vielleicht der ein oder andere sogar erfahren hat, wäre die Einheitlichkeit eines Betriebssystems. Für dieses Beispiel nehmen wir speziell die Betriebssysteme aus aktuellen Smartphones. So ist es in der Menüführung für IOS-Nutzer natürlich und sehr intuitiv, in einer Anwendung, mit einem Tipp auf die obere linke Bildschirmecke wieder eine Ebene zurückzukommen. Für Android-Nutzer liegt diese Funktion unten links. Wenn dann plötzlich diese Funktion in einer App eines Drittherstellers anders aufgebaut ist, bricht es die gefühlte Intuition des Benutzers und er muss sich wieder neu orientieren. Diese Designentscheidung wirkt für den Nutzer schlecht durchdacht sowie aus dem Kontext gerissen. Hier wurden Designanforderungen nicht richtig verstanden, da die Anwendungen nicht richtig in ihrer Laufumgebung angepasst wurde.

2.5 Smartphone-Betriebssysteme

2.6 Anwendungs-Suites

3 Aristocracy and Democracy

Die konzeptionelle Integrität schreibt vor, dass der Entwurf von einem einzigen Kopf oder von einer sehr kleinen Anzahl übereinstimmender Köpfe ausgehen muss. Der Zeitdruck diktiert jedoch, dass der Systemaufbau viele Hände braucht. Es gibt zwei Möglichkeiten, dieses Dilemma zu lösen. Die erste ist eine sorgfältige Arbeitsteilung zwischen Architektur und Implementierung. Die zweite ist die neue Art der Strukturierung von Programmier-Implementierungsteams, die im vorigen Kapitel besprochen wurde. Die Trennung von architektonischem Aufwand und Implementierung ist eine sehr wirksame Methode, um konzeptionelle Integrität bei sehr großen Projekten zu erreichen. Der Architekt eines Systems ist, wie der Architekt eines Gebäudes, der Vertreter des Benutzers. Seine Aufgabe ist es, sein fachliches und technisches Wissen im uneingeschränkten Interesse des Benutzers einzubringen, im Gegensatz zu den Interessen des Verkäufers, des Herstellers usw. Die Architektur muss sorgfältig von der Umsetzung unterschieden werden. Wie Blaauw sagte: "Während die Architektur sagt, was geschieht, sagt die Implementierung, wie es geschieht". Als einfaches Beispiel nennt er eine Uhr, deren Architektur aus dem Zifferblatt, den Zeigern und dem Aufziehknopf besteht. Wenn ein Kind diese Architektur erlernt hat, kann es die Zeit an einer Armbanduhr ebenso leicht ablesen wie an einem Kirchturm. Die Umsetzung und die Verwirklichung beschreiben jedoch, was im Inneren des Gehäuses vor sich geht: die Energieversorgung durch einen von vielen Mechanismen und die Kontrolle der Genauigkeit durch einen von vielen.

4 Wischiwaschi

Die Definition der widerspruchsfreien Anforderungen mit hoher Kohärenz ist schwierig greifbar und macht es problematisch, die Integrität an sich zu messen. Eine gute Alternative, um die Integrität deutlicher zu machen, wäre die Abwesenheit von dieser festzustellen. Für diese Ausarbeitung haben wir uns für den Begriff 'Wischiwaschi' entschieden, da er auch in unserer Quelle so Verwendung findet und uns auch etwas passend scheint. Indikatoren für Wischiwaschi wären:

- Unklare, widersprüchliche oder mehrdeutige Anforderungen

Beispiele für diesen Punkt sind nicht verstandene oder schlecht erklärte User Storys oder Roadmaps.

- Lange Abstimmungsrunden
  Gründe wären zu lange Diskussionen, zu viele Verständnisfragen oder Erklärungen zum Projekt

- Hohe Änderungsraten
  Wenn sich die Anforderungen in einer hohen Frequenz ändern und der aktuelle Entwicklungsstand dementsprechend aktualisiert werden muss. Beispiele wären von Anfang an unklare Anforderungen, die während der Projektlaufzeit spezifiziert werden.

- Explodierende Kosten
  Da die tatsächlichen Kosten die der geplanten Kosten übersteigen ist es möglich, dass die Projektplanung, aufgrund falsch verstandener Anforderungen, schiefgelaufen ist.

- Projektverzug
  Gründe können für ein Projektverzug sein, dass die Aufwandsschätzungen, wegen mangel an Verständnis für das Projekt, einfach schlecht waren.

- Hohe Komplexität
  Es werden im Team viele Fehler gemacht oder das Verstehen des Projektes scheint schwer zu sein. So steigt die Schieflage immer mehr, je mehr Anforderungen an das Projekt gestellt werden.

- Schlechte Performance
  Die Performance ist daher ein Indiz, da das Projekt für die Anwendung ungeeignete oder falsche Werkzeuge nutzt. Mögliche Gründe wären hier zu hohe Änderungsraten, bei denen dann die Werkzeuge nicht optimiert oder ausgetauscht wurden. Auch ein falsches Verständnis vom Projekt könnte eine falsche Annahme der benötigten Mittel herbeiführen.
  Ein weiterer Grund könnte die unnötige Komplexität des Projektes sein. Funktionen, die einander beeinflussen und somit die Performance reduzieren.

- Hohe Lernkurve der Anwendung
  Hohe Lernkurven können passieren aufgrund nicht durchdachtem Design der Anwendung. Ursachen dafür könnten schlecht verstandene Anforderungen, die zu einem schlechten Designkonzept führen.

5 Maßnahmen gegen Wischiwaschi

Um diese Problematiken zu lösen gibt es mehrere Ansatzpunkte, die wir in Folge erklären.

5.1 Anforderungsmanagement

Mit Anforderungsmanagement ist hier die Methode der Aufnahme und Kommunikation der Anforderungen gemeint. Anforderungen müssen klar, widerspruchsfrei und eindeutig vom Auftraggeber formuliert werden. Sie sind die Grundsteine, damit das Projekt später die konzeptionelle Integrität erfüllt. Um möglichst gute Anforderungen zu erhalten, gibt es Kriterien, die beachtet werden müssen:

- Abgestimmt:
  Anforderungen müssen mit allen Stakeholdern abgestimmt werden. Stakeholder müssen die Anforderung als korrekt und notwendig erachten.

- Eindeutig:
  Alle Leser müssen beim Studium der Anforderung zum selben Ergebnis kommen. Eine Mehrfachinterpretation sollte nicht möglich sein. Dies ist aufgrund der Mehrdeutigkeit von Sprache sehr schwierig und benötigt viel Erfahrung und Präzision im Schriftlichen. Dies spricht auch für den Bezug auf ein konkretes Lösungsdesign (zum Beispiel in Form eines Prototyps), denn nur dann können die Anforderungen von allen gleich interpretiert werden.

- Notwendig:
  Anforderungen müssen den Gegebenheiten im Kontext entsprechen. Eine Gegebenheit kann die Erwartung eines Stakeholders sein oder die API-Spezifikation eines externen Dienstes.

- Konsistent:
  Anforderungen müssen sowohl in sich selbst als auch im Vergleich mit anderen Anforderungen konsistent sein. Es darf also keine Widersprüche geben.

- Prüfbar:
  Anforderungen müssen überprüfbar sein, damit man später feststellen kann, ob sie umgesetzt wurden. Zur Überprüfung eignen sich Tests oder Messungen.

- Realisierbar:
  Nicht jede Anforderung ist im geforderten Budget umsetzbar. Dies ist besonders schwierig, wenn bei den Stakeholdern wenig technisches Verständnis vorhanden ist.

- Verfolgbar:
  Der Ursprung der Anforderung muss angegeben werden. Zudem sollte jede Anforderung einen eindeutigen Identifikator haben, damit sie referenziert werden kann.

- Vollständig:
  Anforderungen müssen vollständig sein, d. h. die gewünschte Funktionalität in Gänze beschreiben.

- Verständlich:
  Die Anforderungen müssen von allen Stakeholdern verstanden werden können. Um dies zu erreichen, sollten sie entweder in der Allgemeinsprache formuliert werden oder es müssen Begriffe in einem Glossar genau erklärt werden.

5.2 Werkzeuge/Tools

- Visualisierungen

- Application-Lifecycle-Management-Werkzeuge
  Die Anforderungen müssen möglichst widerspruchsfrei und kohärent bleiben, auch bei Änderungen. Da dies durch viele Informationen und hohe Änderungsfrequenz schwierig ist, können wir uns sogenannte Application-Lifecycle-Management-Werkzeuge (ALM-Werkzeuge) zur Hilfe nehmen. Diese unterstützen die strukturierte Ablage sowie Änderungsprozesse der Informationen gut.

- Taxonomie-Software

- Wiki-Software
  So brauchen manche Entwicklungen, vor allem die mittelgroßen bis großen Projekte, mehrere Seiten an Dokumentationen. Teilweise können diese Dokumentationen für die Mitarbeiter unübersichtlich werden und dazu führen, dass die Softwareentwicklung so ins Stocken kommt. Um dies zu verhindern, muss geschaut werden, dass die wichtigen Informationen auf eine zugängliche Art und Weise dokumentiert als auch

aktualisiert werden, um Fehlinformationen oder Missinterpretationen zu minimieren.

6 Vorteile hoher konzeptioneller Integrität

6.1 Wartbarkeit/Änderbarkeit

Je konsistenter die Anforderungen und je klarer die Ziele und Epics, desto einfacher ist es, Änderungen an den Anforderungen einzubringen, und die entsprechenden Anpassungen an der Software vorzunehmen. Daraus folgt, dass die Software auch länger in Betrieb gehalten werden kann, da sie sich besser anpassen lässt.

6.2 Bedienbarkeit/Erlernbarkeit

Je schlüssiger die Funktionen eines Systems für die Benutzer sind, desto einfacher lassen sie sich auch erlernen. Schon Brooks hat dies behauptet, und es ergibt für uns durchaus Sinn.

6.3 Angepasste Komplexität

Da Ihr System gut durchdacht ist und eine hohe Kohäsion hat, entsteht keine unnötige Komplexität. Das System kann mit einem optimalen Komplexitätsgrad gebaut werden.

6.4 Performance

Unnötige Komplexität in der Source kann Auswirkungen auf die Performance haben. Da keine unnötige Komplexität entstehen muss, ist der Grundstein für eine gute Performance gelegt.

6.5 Planbarkeit

Da die Features und ihre Abbildung auf die Software vom Team gut verstanden werden, sind auch die Aufwandschätzungen gut. Die Planbarkeit nimmt zu.

7 Konzeptionelle Integrität bei Scrum

8 Fazit

Literatur

[1] Frederick P. Brooks. *The mythical man-month – Essays on Software-Engineering.* Addison-Wesley, 1975.

[2] Wikipedia. Gleitkommazahl — wikipedia, die freie enzyklopädie, 2022. [Online; Stand 15. November 2022].

[3] Wikipedia. Integer (datentyp) — wikipedia, die freie enzyklopädie, 2022. [Online; Stand 15. November 2022].