Ausarbeitung Softwarequalität

Software-Qualitätsbewertung mittels Software Metriken

Softwarequalität
Duale Hochschule Baden-Württemberg
Stuttgart

von
Paul Walker und Leon Kampwerth

Matrikelnummer: 3610783, 5722356
Abgabedatum: 22.03.2023

Inhaltsverzeichnis

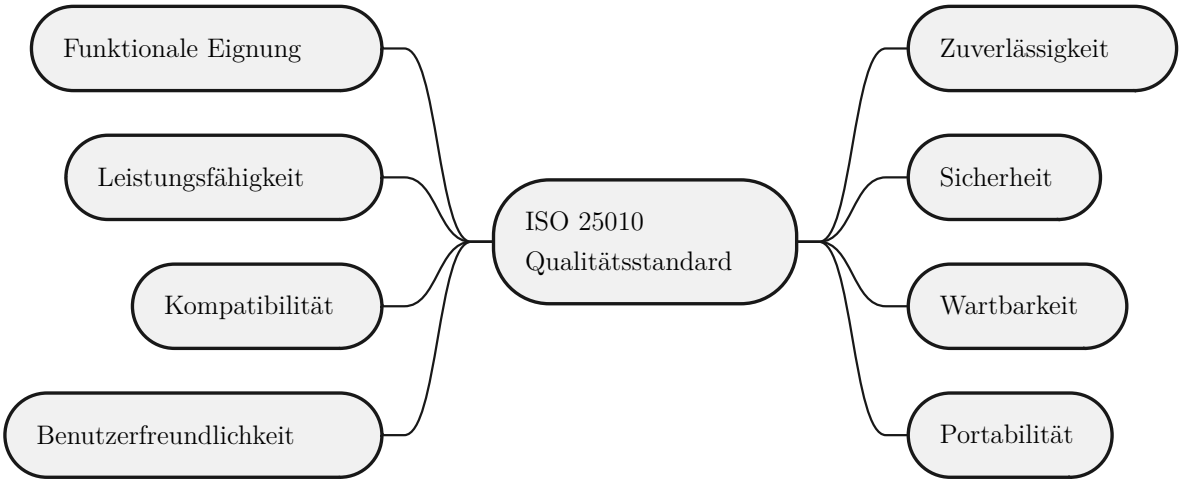

Abbildung 1: Softwarequalität nach ISO 25010[3]

1 Einleitung

Die meisten Menschen arbeiten jeden Tag mit verschiedener Software. In der Arbeit mit Office-Software oder je nach Beruf spezialisierter Software für bestimmte Aufgaben und/oder Maschinen. Zu Hause nutzen viele Menschen Unterhaltungssoftware wie Spiele, oder Videoplattformen. Software ist heutzutage allgegenwärtig und die meisten Menschen, werden auch schon einmal Software schlechter Qualität genutzt haben. Das kann durch abruptes Schließen einer Applikation, unintuitives Design oder langer Ladezeiten für den Nutzer erkennbar sein. Es kann recht einfach sein, schlechte Qualitäten in Software zu erkennen, allerdings lässt sich Softwarequalität so nicht Objektiv vergleichen. Diese Arbeit beschäftigt sich damit Metriken zu finden, mit denen Software Qualität Objektiv gemessen werden kann.

2 Softwarequalität

Um Softwarequalität messen zu können muss zuerst festgelegt werden, was Softwarequalität überhaupt ist. Viele Aspekte von Softwarequalität sind für den Endnutzer gar nicht erkennbar oder sind nur dann erkennbar, wenn dieser Aspekt eine schlechte Qualität hat.

In dieser Arbeit wird die Definition von Softwarequalität nach International Organization for Standardization (ISO) 25010[3] verwendet. In dem ISO 25010 Standard wird ein Produktqualitätsmodell für System und Softwarequalität. Nach diesem Modell wird die Qualität eines Softwaresystems oder Produkts durch acht Charakteristika definiert. Diese Charakteristika werden in Abbildung 1 dargestellt.

Jedes dieser Charakteristika wird weiter in Subcharakteristika aufgeteilt. In dieser Arbeit wird der Fokus auf zwei bestimmte Charakteristika gesetzt und für jedes eine Metrik vorgestellt, mit der die Qualität der Charakteristik gemessen werden kann.

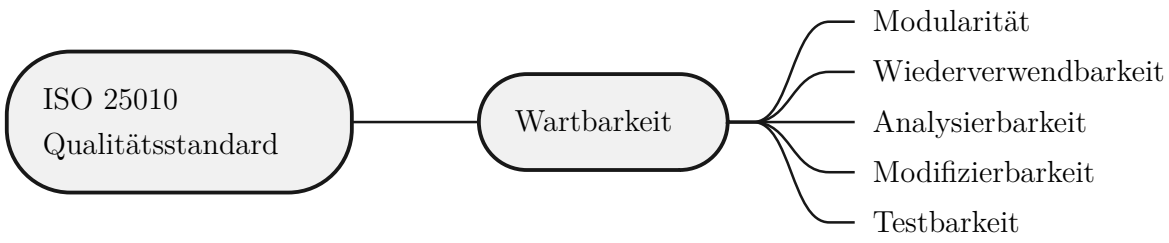

Abbildung 2: Wartbarkeit nach ISO 25010[3]

2.1 Wartbarkeit

Der Grad der Wartbarkeit ist nach ISO, wie effektiv und effizient das Produkt oder System verändert werden kann [3]. Diese Veränderungen können Korrekturen, Verbesserungen oder Anpassungen der Software auf Änderungen des Einsatzgebiets oder auf Änderungen der Anforderungen sein [3]. Die Wartbarkeit bezieht sich auch auf die Installation von Updates [3]. Die Wartbarkeit ist nach ISO in weitere Subcharakteristika aufgeteilt. Diese Subcharakteristika sind in Abbildung 2 dargestellt.

2.1.1 Modularität

Die Modularität wird daran gemessen, inwieweit das System oder Programm aus einzelnen Teilen besteht und Änderungen an einem Teil der Software andere Teile nur minimal oder gar nicht beeinflussen [3]. Module sind Softwareattribute, die Software in unabhängige Einzelteile trennen [4].

2.1.2 Wiederverwendbarkeit

Die Wiederverwendbarkeit ist der Grad, zu dem ein Teil des Systems in mehr als einem System oder beim Erstellen von weiteren Teilen des Systems verwendet werden kann [3]. Die Wiederverwendung bezieht sich dabei nicht nur auf Software, sondern auch auf Dokumentation, Software-Design, Applikationstests und weitere Teile des Systems [2].

2.1.3 Analysierbarkeit

Wie effektiv und effizient es möglich ist, den Einfluss auf ein Produkt oder System abzuschätzen, wenn ein oder mehrere Teile verändert werden, ist durch die Analysierbarkeit definiert [3]. Außerdem wie gut der Ursprung von Fehlern im Produkt oder System diagnostiziert werden kann [3].

2.1.4 Modifizierbarkeit

Die Modifizierbarkeit ist ein Maßstab dafür wie effektiv und effizient sich ein Produkt oder System ändern lässt, ohne dabei Defekte einzuführen oder die Produktqualität zu verschlechtern [3].

Die Modifizierbarkeit kann durch die *Modularität* und die *Analysierbarkeit* beeinflusst werden [3].

2.1.5 Testbarkeit

Der Grad der Testbarkeit ist, wie effektiv und effizient Testkriterien für ein Produkt oder System definiert werden können und wie effektiv und effizient Tests für das Produkt oder System durchgeführt werden können [3]. Zusätzlich zur Erstellung der Testkriterien und der Durchführung der Tests, spielt es auch eine Rolle wie gut objektive und messbare Tests erstellt werden können, die feststellen, ob ein bestimmtes Kriterium erfüllt wurde [4].

2.2 TODO @Leon Charakteristika X

3 Software Metriken

3.1 Wartbarkeit

Viele Charakteristika der Wartbarkeit lassen sich über die Dauer oder Effizienz von Wartungsaufgaben messen. Es kann beispielsweise die Zeit gemessen werden, die benötigt wird, um eine Änderung an der Software durchzuführen, um damit die Modifizierbarkeit zu charakterisieren [1].

Diese Metriken lassen sich nur bei konkreten Wartungsaufgaben messen. In dieser Arbeit soll allerdings die Wartbarkeit einer neu entwickelten Softwarekomponente, die noch keine Wartungsarbeiten benötigt, gemessen werden. Zudem sind solche Zeitmetriken von der jeweiligen Entwicklererfahrung abhängig [1]. Aus diesen Gründen werden solche Metriken in dieser Arbeit nicht verwendet.

Wichtig für die hier präsentierten Metriken ist der Begriff der Einheit und die Einheitengröße. Eine Einheit ist das kleinste ausführbare Stück Code [1]. Je nach Programmiersprache kann eine Einheit eine Funktion, eine Klasse oder aber das gesamte Programm sein [1]. Hier wird eine Klasse als Einheit angenommen.

3.1.1 Modularität

Eine einfache Metrik für die Modularität ist die Modulanzahl in dem betrachteten Teil der Software. Nachteil dieser Metrik ist, dass sie nicht aussagt, inwieweit Änderungen an einem Modul die anderen Module beeinflussen. Nach ISO 25010 sollen Änderungen an einem Modul, die anderen Module möglichst wenig beeinflussen [3].

Andere Metriken sind die Anzahl der Klassen und die Klassengröße. Das gibt auch eine gewisse objektive Aussage über die Modularität. Für eine hohe Modularität ist eine geringe Kopplung der Klassen und hohe Kohäsion, also hohe Zusammengehörigkeit, innerhalb der Klassen nötig um zu gewährleisten, dass Änderungen an einem Modul, die anderen Module minimal oder gar nicht beeinflussen. Viele kleine Klassen sprechen für eine hohe Kohäsion, da in einer kleinen Klasse nur wenig Funktionalität, die dann aber

stark zusammenhängt, implementiert werden kann. Daher kann bei vielen kleinen Klassen von einer höheren möglichen Modularität als bei wenigen großen Klassen ausgegangen werden. Allerdings können auch diesen Metriken keine direkte Aussage über die Modularität liefern.

In dieser Arbeit werden alle der drei Metriken verwendet, um die Modularität der entwickelten Softwarekomponenten zu messen und eins zu eins verrechnet, um einen Wert für die Modularität zu bekommen.

3.1.2 Wiederverwendbarkeit

Die Wiederverwendbarkeit lässt sich nicht direkt messen, allerdings kann gemessen werden wie oft Softwarekomponenten tatsächlich wiederverwendet werden, also die Wiederverwendung. Bei hoher Wiederverwendung muss auch die Wiederverwendbarkeit hoch sein. Allerdings muss eine niedrige Wiederverwendung nicht heißen, dass die Softwarekomponenten nicht wiederverwendbar sind. Trotzdem ist eine niedrige Wiederverwendung ein starker Indikator für niedrige Wiederverwendbarkeit, da Softwarekomponenten, die Wiederverwendbar sind, oft auch wiederverwendet werden.

Die Wiederverwendung lässt sich direkt messen. Dafür muss gezählt werden, wie oft der zu analysierende Teil der Software an anderen Stellen in der Software verwendet wird. Bei Klassen kann beispielsweise gezählt werden an wie vielen Stellen diese instanziiert werden, oder auf sie zugegriffen wird.

Eine weitere Metrik für die Wiederverwendung ist die Code-Duplikation und die Einheitengröße. Wenn hohe Code-Duplikation vorhanden ist, ist die Wiederverwendung in der Regel gering, da duplizierter Code oft wiederverwendbarer implementiert werden kann. Durch Extrahieren des duplizierten Codes in Funktionen oder Klassen, kann die Wiederverwendung beispielsweise erhöht werden. Bei geringer Code-Duplikation kann davon ausgegangen werden, dass eine hohe Wiederverwendung des Codes gegeben ist. Für eine hohe Wiederverwendbarkeit ist es wichtig, dass Einheiten nur wenige Aufgaben haben. Meistens werden nur einzelne Aufgaben an anderer Stelle benötigt, weshalb es wichtig ist, dass Einheiten nur eine oder wenige Aufgaben haben, um eine Einheit wiederzuverwenden. Da Einheiten die nur eine oder wenige Aufgaben haben, in der Regel recht klein sind, deuten geringe Einheitengrößen eine höhere Wiederverwendbarkeit an. Code-Duplikation wird in duplizierten Code-Zeilen gemessen und bewertet und Einheitengröße wird in durchschnittlichen Code-Zeilen pro Einheit gemessen und bewertet. Die Ergebnisse werden mit der Gewichtung eins zu eins verrechnet, um eine Metrik zur Wiederverwendbarkeit zu bekommen.

3.1.3 Analysierbarkeit

Die Analysierbarkeit einer Softwarekomponente wird von verschiedene Faktoren beeinflusst. Aus einigen dieser Faktoren lässt sich eine Metrik zu Analysierbarkeit erstellen. Es eignen sich besonders Faktoren, die einfach messbar sind. Hier werden nach [1] Code-Volumen, Komplexität der Einheiten, Code-Duplikation, Einheitengröße und Unit-Testing als einfach messbare Faktoren verwendet.

3.1.4 Modifizierbarkeit

In [1] wird die Komplexität der Einheiten und die Code-Duplikation als Metrik für die Modifizierbarkeit genannt. Code-Duplikation verringert die Modifizierbarkeit, da an Stellen mit dupliziertem Code, alle Teile des duplizierten Codes gleichermaßen geändert werden müssen. Das erhöht den Modifikations-Aufwand stark und erhöht die Chance, dass bei Modifikationen Teile des duplizierten Codes vergessen werden zu ändern und so Fehler im Programm entstehen. Die Komplexität der Einheiten ist wichtig, da höhere Komplexität von Einheiten dafür sorgt, dass es schwieriger ist diese Einheiten zu modifizieren, ohne Fehler in der Software zu erzeugen.

3.1.5 Testbarkeit

Die Testbarkeit von Software hängt unter anderem von der Komplexität der Einheiten, der Einheitengröße und der Zahl der Unit-Tests ab [1]. Der Einfluss der Unit-Tests auf die Testbarkeit ist einfach zu erkennen. Viele Unit-Tests sind ein Indikator für gute Testbarkeit, denn das bedeutet, dass die Software im allgemeinen testbar ist. Kleine Einheitengröße sorgt auch für eine bessere Testbarkeit, da es einfacher ist für kleine Einheiten, die weniger Aufgaben haben, Testfälle zu spezifizieren. Für kleinere Einheiten müssen in der Regel auch weniger Testfälle spezifiziert werden als bei großen Einheiten um eine hohe Testabdeckung und damit eine höhere Testbarkeit zu erreichen. Bei großen Einheiten ist auch eine höhere Zahl von Seiteneffekte zu erwarten, die das Testen erschweren. Gewisse Seiteneffekte lassen sich auch in kleinen Einheiten nicht vermeiden, allerdings ist mit weniger Seiteneffekten als bei großen Einheiten zu rechnen. Kleinere Einheiten haben weniger Aufgaben und benötigen damit auch weniger Seiteneffekte. Um eine Einheit testen zu können, müssen alle Seiteneffekte isoliert und kontrolliert werden. Ist das nicht der Fall, kann kein sinnvoller Test spezifiziert werden. Der Aufwand eine Einheit zu testen steigt stark an, je mehr Seiteneffekte isoliert und kontrolliert werden müssen. Außerdem lassen sich einige Seiteneffekte speziell im Oberflächenbereich teilweise gar nicht isolieren und kontrollieren, was dafür sorgt, dass die ganze Einheit nicht getestet werden kann. Die Komplexität der Einheiten spielt auch eine Rolle, denn es ist einfacher für weniger komplexe Einheiten Testfälle zu spezifizieren als für sehr komplexe Einheiten.

3.2 TODO @Leon Metrik X

4 Ausblick

Literatur

[1] Ilja Heitlager, Tobias Kuipers und Joost Visser. „A Practical Model for Measuring Maintainability“. In: *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*. IEEE. 2007, S. 30–39. DOI: 10.1109/QUATIC.2007.8.

[2] „IEEE Standard for Information Technology–System and Software Life Cycle Processes–Reuse Processes“. In: *IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999)* (2010), S. 1–51. DOI: 10.1109/IEEESTD.2010.5551093.

[3] ISO 25010. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Standard. Geneva, CH: International Organization for Standardization, 2011.

[4] „ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary“. In: *ISO/IEC/IEEE 24765:2017(E)* (2017), S. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.