Advanced Seminar SoSe 2022
Nachhaltigkeit in der Sotwareentwicklung

Wie kann das Handlungsprinzip der Nachhaltigkeit bei Entwicklung und Betrieb einer Web-Anwendung zur Verwaltung und Buchung von Räumen am Campus Gummersbach der TH Köln angewandt werden?

Projektvorstellung

Im Rahmen eines hochschulinternen Projekts der TH Köln am Campus Gummersbach sollen Raumbuchungen für Studierende und Lehrende konzeptionell hinterfragt und digitalisiert werden. Dazu wird eine Online-Plattform entwickelt, die unkomplizierte Raumverwaltungsprozesse und Buchungsprozesse implementiert und benutzerfreundlich aufbereitet. Nach der Anforderungsermittlung soll zusätzlich ein Architekturentwurf des Systems und erste Prototypen entwickelt werden. Bei diesen Entwürfen soll eine hohe Gewichtung auf die Nachhaltigkeit des Systems gelegt werden. In der Vergangenheit hat sich an der TH Köln oft gezeigt, dass nicht-nachhaltig geplante Systeme für die Systempflegenden und -nutzenden einen nicht zu vernachlässigenden Mehraufwand erzeugen und damit eine erhöhte Unzufriedenheit einhergeht. Ebenfalls entsteht das Problem, dass Software nach einem kurzen Nutzungszeitraum nur noch schwierig an neue Anforderungen angepasst werden kann. Aus diesem Grund wird bei der Planung und Implementierung eine hohe Gewichtung auf Nachhaltigkeit im ökonomischen Sinn gelegt. Daher wird in dieser Ausarbeitung der Begriff der Nachhaltigkeit für dieses Projekt genau definiert, technische und organisatorische Maßnahmen zur Erreichung dieser definiert und Messwerte bestimmt, um die Wirksamkeit der Maßnahmen auch im späteren Verlauf des Software-Lebenszyklus zu gewährleisten.

Nachhaltigkeit

Der Begriff Nachhaltigkeit entsprang ursprünglich der Forstwirtschaft und besagte, dass in einem Wald nur so viele Bäume gefällt werden dürfen, wie in absehbarer Zeit nachwachsen können. Wer nachhaltig handelt, plant voraus und sichert Bedürfnisbefriedigung auch für die Zukunft. Heutzutage breitet sich dieses Handlungsprinzip über zahlreiche andere Anwendungsfelder aus - darunter Produktionen aller Art, aber auch Wissenschaft und Politik. Auch in der Softwareentwicklung wird vermehrt auf Nachhaltigkeit geachtet.

Benedikt Engel, Tim Loges, Yannic Brügger

Scope der Arbeit

Der Nachhaltigkeitsbegriff im Kontext der Softwareentwicklung kann aus zwei verschiedenen Perspektiven betrachtet werden.

Die Software selbst ist nachhaltig, indem in der Entwicklungs- und Wartungsphase (Nachhaltigkeit **IN** der Software). Der Fokus liegt auf der nachhaltigen Entwicklung von Software.

Die Software wird zum erreichen von Nachhaltigkeitszielen entwickelt und eingesetzt. Beispiel: App zum Messen des eigenen CO2 Austoßes (Nachhaltigkeit **DURCH** Software). Der Fokus liegt auf dem Nachhaltigen Nutzen der Software [1].

Drei Perspektiven auf Nachhaltigkeit

Ökonomische Nachhaltigkeit (economic Sustainability) aus Perspektive der Wirtschaft (Economy) beschreibt, wie der Lebenszyklus von Software das Investment von Stakeholdern schützt, Geschäftswerte erzielt, Risiken minimiert und Vermögenswerte auf lange sicht erhält.

Ökologische Nachhaltigkeit (ecological Sustainability) aus Perspektive der Umwelt (Environment) beschreibt, wie Entwicklung, Wartung und Benutzung von Software sich auf den Energieverbrauch und den Verbrauch von anderen Ressourcen auswirkt.

Soziale Nachhaltigkeit (social Sustainability) aus Perspektive der Gesellschaft (Society) beschreibt, wie Softwareentwicklung und -wartung die Lebensqualität der Entwickler:innen beeinflusst [1].

Scope der Arbeit: *Nachhaltigkeit in der Softwareentwicklung mit Fokus auf ökonomische Nachhaltigkeit*

Einflüsse auf die Nachhaltigkeit

Untersuchung der Studie *"An empirical analysis of the impact of software development problem factors on software maintainability"*. Denn, Wartbarkeit von Software hat einen starken Einfluss auf die Nachhaltigkeit dieser.

Zwei Kernfragen daraus sind relevant:
Wie beeinflussen Software-Process-Improvements (SPI) die Entwicklungsphase und die darauf folgende Wartungsphase?
Was sind die 10 Hauptfaktoren, die die spätere Wartbarkeit maßgeblich beeinflußen?

Empirische Untersuchung in 5 Dimensionen:
- Dokumentations-Probleme
- Programmier-Qualitäts-Probleme
- System-Anforderungs-Probleme
- Personelle-Probleme
- Prozess-Management-Probleme

Erkenntnisse der Studie:
Projekte mit hoher Wartbarkeit sind nicht nur kleine Projekte.
Die Erfahrung der Entwickler:innen muss nicht im Zusammenhang mit der Wartbarkeit eines Projektes stehen.
Die Probleme in der Entwicklungsphase unterscheiden sich von denen in der Wartungsphase. Aber, die Entwicklungsphase hat einen Einfluss auf die später Wartungsphase [2].

Rank	Software development problem factors	Abbreviation	Mean values (n = 137)
1	Inadequacy of source code comments	PGM2	4.05
2	Documentation is obscure or untrustworthy	DOC1	4.04
3	Changes are not adequately documented	DOC4	3.87
4	Lack of traceability	DOC3	3.85
5	Lack of adherence to programming standards	PGM1	3.69
6	Lack of integrity/consistency	DOC5	3.66
7	Continually changing system requirements	SYS5	3.58
8	Frequent turnover within the project team	PER1	3.55
9	Improper usage of programming techniques	PGM5	3.51
10	Lack of consideration for software quality requirements	SYS4	3.51

Tabelle 1: Die 10 größten Einflussfaktoren auf die Wartbarkeit [2].

Nachhaltigkeit durch Architektur

Die Architektur einer Software wirkt sich direkt auf die Wartbarkeit und dadurch auf die Nachhaltigkeit von Software aus. Jede Komponente verfügt über Funktionen, Verantwortlichkeiten, und Abhängigkeiten zu anderen Systemen/Komponenten. Bei der Entwicklung einer Architektur können verschiedene Bad Smells auftreten, die potentiell die Qualität/Nachhaltigkeit von Software kompromitieren [3].

Allgemein lässt sich sagen, dass sich Modularität positiv auf Austauschbarkeit, Erweiterbarkeit und Wiederverwendebarkeit auswirkt. Ein aktuelle Trend in der Softwarearchitektur, der diese Idee verfolgt, ist das Architekturprinzip der Microservices. Hierbei werden verschiedene Bereiche der Verantwortlichkeiten absichtlich voneinander abgelöst, sodass statt einem großen Programm (Monolith) mehrere kleine - in sich abgeschlossene - Programme (Microservices) entstehen. Dieses Prinzip wird verwendet, um im hier beschreibenen Projekt, austauschbare und wiederverwendbare Komponenten zu erhalten [siehe Anhang].

Nachhaltigkeit durch Technologieauswahl

Welche Technologien zum erreichen einer Architektur verwendet werden, hat einen Einfluss auf die Haltbarkeit eines Systems. Werden Technologien verwendet, die schon bald von Herstellern aus dem Support genommen werden (Deprectated Software), muss das System voraussichtlich früher gewartet werden. Auf der anderen Seite sollte eine Technologie ausgereift und erprobt genug sein, um lange Verlässlichkeit sicherzustellen. Hierzu helfen Tech-Radare, die Technologien entsprechend Kategorisieren [siehe Anhang].

Nachhaltigkeit durch Dokumentation

Ein weiterer Aspekt, der in [3] als einer der kräftigsten Einflussfaktoren bestimmt wurde, ist die Dokumentation von Softwaresystemen. Es werden verschiedene Arten der Dokumentation einzeln Adressiert. Für jede Problematik ist eine Lösung aufgelistet, die für den Einsatz im Projekt vorgeschlagen wird.

Kommentare im Code können der Dokumentation dienen, werden aber häufig unverwaltet. Oft sind sie ungenau, zu zahlreich und verwirren dadurch mehr, als sie der Dokumentation dienen. Es wird empfohlen Kommentare möglichst zu vermeiden und stattdessen aussagekräftigen Code zu produzieren [4].

Änderungen am Quelltext müssen gut dokumentiert sein. Jederzeit muss jede Änderung nachvollzogen werden können. Dabei hilft ein Zusammenspiel aus Issues, Pull Requests und Commit-Messages.

Um Konsistenz in der Dokumentation zu wahren, ist es sinnvoll auf bereits bestehende Standards zurückzugreifen oder (falls nötig) eigene Standards zu definieren und zu dokumentieren. Konventionen können beispielsweise *Conventional Commits* oder bestimmte Programmier-Paradigmen wie *SOLID-Prinzipien* und *Pure-Functions* sein. Dokumentation geht über den Quelltext hinaus. Auch Architekturentscheidungen müssen Dokumentiert werden. Dafür empfiehlt sich der Standard *Architecture Decision Records*.

Nachhaltigkeit durch Tests

Um eine nachhaltige, dauerhaft stablie Anwendung zu garantieren sollte ein Testing-Framework verwendet werden und für alle Funktionen Tests geschrieben werden, welche die richtige Funktionsweise nach der Entwicklung oder des Refactorings überprüfen. Durch diese Tests lassen sich schon vor dem Deployment fehlerhafte Strukturen erkennen und somit gar nicht erst deployen.
Zudem können Tests auch als Dokumentation des Quellcodes genutzt werden, da ein Test die richtige Funktionsweise einer Funktion nutzt und darstellt [5].

Messbarkeit und Kennzahlen

Um den Erfolg der Nachhaltigkeit messbar zu machen, können die RAMP-Prinzipen gemessen werden. Diese bestehen aus den folgenden nicht-funktionalen Anforderungen:

Verlässlichkeit (eng. Reliability)
Die Verlässlichkeit beschreibt, wie hoch die Wahrscheinlichkeit ist das keine Fehler während der Laufzeit auftreten [6].
Eine Möglichkeit dies zu messen ist hierbei MTTF (Mean Time to Failure), also das Zeitintervall welches vergeht zwischen dem Zeitpunkt an dem ein Fehler aufgetreten ist und dem nächsten Fehler. Je höher dieser Wert ist desto besser.

Verfügbarkeit (eng. Availability)
Zwei Sichtweisen möglich: Verfügbarkeit der Anwendung im allgemeinen oder Verfügbarkeit durch erfolgreiche Anfragen an die Anwendung.

allg. Anwendung: $\frac{\text{Uptime}}{\text{Uptime + Downtime}}$

Anfragen: $\frac{\text{Anz. erfolgreiche Anfragen}}{\text{Anz. erfolgreiche + Anz. nicht erfolgreiche Anfragen}}$

[7]

Wartbarkeit (eng. Maintainablitiy)
Durch Einsatz von Technologien die als effizient, stabil und zukunftsorientiert angesehen werden, gute Dokumentation, möglichst kleinem Technologie-Stack (Lean) um komplexität vorzubeugen.
Schwierig zu messen, da Dauer auch vom Umfang der Aufgabe abhängt, aber möglich durch:

$$\frac{\text{tatsächlicher Aufwand der Umsetzung}}{\text{geplanter Aufwand}}$$

Performanz (eng. Performance)
Die Performanz einer Anwendung kann durch viele verschiedene Kennzahlen dargestellt werden, unteranderm durch die Antwortzeit, also die Zeit welche zwischen Anfrage und vollständiger Antwort des Servers vergeht, aber auch durch die Anzahl von gleichzeitig möglichen Anfragen pro Sekunde [6].

Problem: Kennzahlen lassen sich nur schwer vergleichen, da jedes Softwareprojekt unterschiedlich ist, auch bei gleichem Thema/Auftrag kann ein *nicht* nachhaltig-entwickeltes Softwareprojekt besser abschneiden als ein nachhaltig-entwickeltes.

Ausblick

In dieser Arbeit wurde ausschließlich *ein* Teil von Nachhaltigkeit in *einem* Softwareprojekt betrachtet (siehe Scope der Arbeit). Andere Arbeiten haben bereits allgemeine Modelle vorgestellt, die weitere Aspekte von Nachhaltigkeit in der Softwareentwicklung berücksichtigen [8]. Eine Ausweitung des Nachhaltigkeitsverständnisses auf dieses Projekt, könnte in der Zukunft an Relevanz gewinnen.

Um die Wirksamkeit der hier beschriebenen Maßnahmen zu bestimmen, muss das Projekt einige Zeit nach Produktivnahme auf Nachhaltigkeit untersucht werden. Dafür wurden entsprechende Kennzahlen definiert.

Quellen:
[1] C. Calero, M. Á ngeles Moraga und F. Garc ́ıa, "Software, Sustainability, and UN Sustainable Deve- lopment Goals," IT Professional, Jg. 24, Nr. 1, S. 41–48, 2022. doi: 10.1109/MITP.2021.3117344.
[2] J. Chen, S. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability", Journal of Systtems and Software, Vol. 82, Issue 6, S.981-992. doi: https://doi.org/10.1016/j.jss.2008.12.036.
[3] D. M. Le, C. Carrillo, R. Capilla and N. Medvidovic, "Relating Architectural Decay and Sustainability of Software Systems," 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2016, pp. 178-181, doi: 10.1109/WICSA.2016.15.
[4] R. C. Martin, Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code ; [Kommentare, Formatierung, Strukturierung ; Fehler-Handling und Unit-Tests ; zahlreiche Fallstudien, Best practices, Heuristiken und Code smells], Dt. Ausg., 1. Aufl. [Heidelberg]: mitp, 2009.
[5] S.Crouch; Software Sustainablitiy Institute - University of Edinburgh (o. Datum). Testing your software [Online]. Adresse: https://software.ac.uk/resources/guides/testing-your-software (Zuletzt aufgerufen: 04.07.2022).
[6] Malkawi, M.I. The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP). Hum. Cent. Comput. Inf. Sci. 3, 22 (2013). https://doi.org/10.1186/2192-1962-3-22
[7] A. Ross, A. Hilton, M. Brown und D. Rensin m(2017). Available . . . or not? That is the question—CRE life lessons [Online]. Adresse: https://cloud.google.com/blog/products/ gcp/available-or-not-that-is-the-question-cre-life-lessons (Zuletzt aufgerufen: 30.06.22).
[8] P. Lago, "Architecture Design Decision Maps for Software Sustainability," 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), 2019, pp. 61-64, doi: 10.1109/ICSE-SEIS.2019.00015.

Technology
Arts Sciences
TH Köln