[KMR20] J. C. Kirchhof, J. Michael, B. Rumpe:
Softwarequalität in Energieprojekten.
In: ENERGIEWENDEBAUEN - Forschungserkenntnisse von der Komponente bis zum Quartier, Fraunhofer IRB Verlag, Stuttgart, July 2020.
https://www.se-rwth.de/publications/

9.8 Softwarequalität in Energieprojekten

9.8.1 Einleitung

Aus Sicht des Software Engineering ergeben sich in der Forschung zu energieoptimierten Gebäuden und Quartieren eine Vielzahl an Fragestellungen. Im Gegensatz zu vorhergehenden Betrachtungen zu softwaretechnischen Herausforderungen für Infrastrukturen zur Ermöglichung und zum Betrieb energieeffizienter Städte [1] fokussieren wir uns im Folgenden auf die Qualität der entstehenden Softwareprodukte und Verbesserung ebendieser.

Zwischen der Entwicklung von Demonstratoren und real verwendeter Softwareprodukten bestehen in der Softwareentwicklung große Unterschiede, die sich insbesondere auf ntwicklungszeit und damit die Kosten solcher Produkte auswirken. Betrachtet man beispielsweise die „Technology Readiness Level (TRL)“, so erreichen viele Projekte TRL 6 oder 7, also Demonstratoren in echter Umgebung oder Prototyp im Einsatz. Daher wird die Software in solchen Projekten oft auch nur für diese Prototyp-Ebene entwickelt. An Systeme in realen Umgebungen, die für mehrere Jahre benutzbar und weiterentwickelbar sein sollen, bestehen jedoch deutlich höhere Anforderungen hinsichtlich der Qualität auf unterschiedlichen Ebenen, z. B. der Zuverlässigkeit, Benutzerfreundlichkeit, Änderbarkeit, Performanz und Wiederverwendbarkeit.

Es stellt sich daher die Frage, welche Designprinzipien für die Entwicklung von qualitativ hochwertiger Software in Energieprojekten relevant und hilfreich sind.

Bei Software entstehen gegenüber herkömmlichen Produktionsprozessen keine Qualitätsmängel durch die Massenproduktion, die sich dann nur selektiv in einem oder einem anderen Produkt widerspiegeln. Bei Software sind Qualitätsmängel in der Entwicklung begründet und ziehen sich über die gesamte Software hinweg. Die Erfahrung hat gezeigt, dass die Qualität der Entwicklungsprozesse erheblichen Einfluss auf die Qualität der Software hat [2]. Neben einer guten Strukturierung der Prozesse (z. B. durch Vorgehensmodelle oder Qualitätsmanagement Standards wie CMMI, SPICE oder die ISO 9000 Familie) sind für Energieprojekte insbesondere Themengebiete, wie die gute Handhabbarkeit der Benutzeroberflächen, die Anpassbarkeit der Systeme, adäquate Softwarearchitekturen sowie die Möglichkeit, auf Änderungen reagieren zu können, relevant. Wir behandeln hier daher eine Reihe von Designprinzipien, deren Einhaltung zu einer besseren Qualität in Energieprojekten führen und zeigen dazu beispielhafte Anwendungen.

9.8.2 Adäquate Softwarearchitekturen

Zentrale Aufgabe in vielen Energieprojekten ist das Sammeln und Auswerten von Daten und darauf basierend das Beeinflussen der Umgebung. Klassischerweise werden dazu Sensoren, die Daten über ihre Umgebung sammeln, und Aktuatoren, die ihre Umwelt beeinflussen können, mit einem oder mehreren Servern verbunden. Aus Kostengründen werden diese Server mehr und mehr von Cloudlösungen verdrängt, bei denen Ressourcen auf Abruf bereitgestellt und damit dynamisch an den tatsäch-

lichen Bedarf angepasst werden können. Das bedingt mehrere Designprinzipien für die Entwicklung von Software Architekturen.
Um die Ressourcen der Cloud effizient zu nutzen, sollte die Software in kleine, voneinander unabhängige Komponenten, sogenannte Microservices [3], aufgeteilt werden. So können auch einzelne Teile der Anwendung auf Abruf benutzt werden, ohne dass nicht benötigte Teile der Anwendung Ressourcen verschwenden. Die Unabhängigkeit der Komponenten erlaubt es, bei hohem Datenaufkommen zu Stoßzeiten benötigte Komponenten durch die Cloud zu replizieren und damit eine Nichtverfügbarkeit des Systems wegen Überlastung zu verhindern. Dadurch, dass die Komponenten nur über klar definierte Schnittstellen Nachrichten miteinander austauschen, können sie leicht in anderen Kontexten wiederverwendet werden. Ein weiterer Vorteil ist, dass die einzelnen Services parallel von verschiedenen Teams entwickelt und getestet werden können, solange für jede Komponente und deren Schnittstellen eine klare Spezifikation existiert.
Um das Ziel einer Architektur mit unabhängigen, leicht austauschbaren, wiederverwendbaren Komponenten zu erreichen, sollten auch bei der Konzeption bzw. Komposition der Komponenten Best Practices beachtet werden. Robert C. Martin ordnet die Komponenten in seinem Leitfaden zur „Clean Architecture“ [4] dazu in konzentrischen Kreisen an, wobei Abhängigkeiten zwischen Komponenten immer nur von einem äußeren Kreis auf einen inneren Kreis verweisen dürfen, aber niemals umgekehrt. Dadurch ist sichergestellt, dass die Anwendungslogik, die sich in den inneren Kreisen befindet, nicht von externen Faktoren wie Frameworks oder GUIs abhängt, die sich in den äußeren Kreisen befinden.

9.8.3 Agilität im Entwicklungsprozess

Die Energiebranche obliegt, wie auch viele andere Anwendungsgebiete, von innerhalb eines Projektes, z. B. Kunden oder Stakeholdern getriebenen Änderungen der Anforderungen an Softwaresysteme, aber auch von extern getriebenen, z. B. durch Änderungen von Gesetzen oder politischen Gegebenheiten. Die Entwicklungsprozesse müssen daher agil auf solche Änderungsbegehren reagieren können. Neben agilen Vorgehensmustern wie Scrum oder Kanban müssen auch die technischen Prozesse und Werkzeuge auf kontinuierliche Änderungen reagieren können und diese unterstützen. Insbesondere Modell-getriebene Ansätze haben sich hier in den letzten Jahren bspw. für die Oberflächengestaltung oder die Testautomatisierung als praktikabel erwiesen [5].

9.8.3.1 Anpassbarkeit und Benutzerfreundlichkeit der Oberflächen

Die Gestaltung von gut bedienbaren und attraktiven graphischen Benutzeroberflächen (GUIs) ist insbesondere für menschenzentrierte Anwendungen relevant, in denen z. B. Bewohner oder Betreiber sowohl Informationen dargestellt bekommen als auch Interaktionen mit einem System durchführen müssen. Eine Möglichkeit, um rasch vorzeigbare Prototypen zu gestalten und diese GUIs anpassbar zu gestalten, ist die Verwendung von Code Genera-

toren, die aus Modellen GUIs (und mehr) generieren können [6]. Abb. 9.8-1 zeigt die Grundprinzipien dieses Prozesses: Aus Modellen, die verwendete Daten sowie darzustellende Oberflächen beschreiben, wird durch Verwendung von Templates für unterschiedliche Zielsprachen eine Datenbank, die Kommunikationsinfrastruktur sowie Backend und Frontend eines Client-Server Systems erstellt. In den GUI-Modellen können verschiedene Darstellungen für unterschiedliche Benutzergruppen definiert werden, die dann entsprechend im Frontend angezeigt werden (siehe Abb. 9.8-1 rechts). Kombiniert man diese Ansätze noch mit Prinzipien von Assistenzsystemen [7], so kann man zudem Benutzerführungen durch die Systeme sowie Unterstützungsfunktionen für die Nutzung der zur Verfügung gestellten Programmfunktionen einfach realisieren. Modell-getriebene Ansätze ermöglichen es, (1) rasch Demonstratoren für Benutzer zur Verfügung zu stellen, die dann weiterentwickelt werden können und nicht weggeworfen werden müssen und (2) schnell auf Änderungswünsche der Benutzer in den Oberflächen reagieren zu können.

Grundsätzlich ist es möglich, solche Code-Generierungsansätze auch auf andere Systemarchitekturen oder Programmiersprachen zu übertragen, in denen stark repetitive Code-Teile vorhanden sind oder die Generierungsmechanismen für andere Anwendungszwecke wie z. B. die Analyse oder das Testen von Codes einzusetzen.

9.8.3.2 Test und Automatisierung

Komplexe Systeme und fehleranfällige Programmiersprachen sind auch in der Energieforschung weit verbreitet. Zudem bergen agile Entwicklungsprozesse das Risiko, dass die Qualität der entwickelten Software darunter leidet. Aus diesen Gründen ist das systematische Testen von Softwaresystemen ein wichtiger Bestandteil des Entwicklungsprozes-

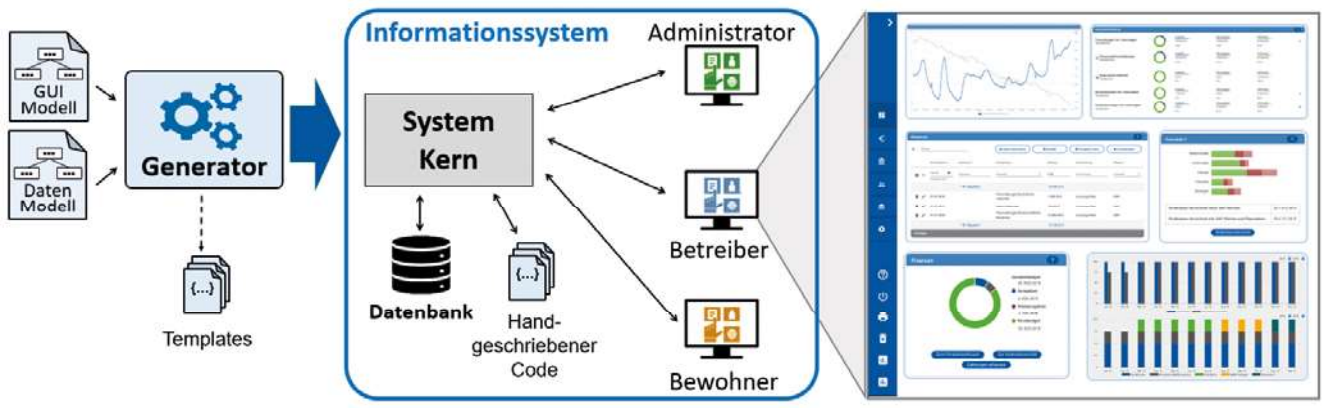

Abb. 9.8-1: Grundprinzip der Generierung von Oberflächen für ein Informationssystem

ses [8]. In typischen Projekten umfasst die Testzeit 40–50 % der Gesamtentwicklungszeit [9], da viele Ebenen getestet werden müssen: es gibt Komponenten-, Integrations- und Systemtests bis hin zu Stress- und Abnahmetests.
Durch ein strukturiertes, methodisches Vorgehen, einer guten Testplanung und einem hohen Automatisierungsgrad sind hier sowohl höhere Testabdeckungen wie auch zeitliche Einsparungen möglich [8]. Eingesetzte Methoden sind hier beispielsweise toolunterstütze gegenseitige Code Reviews, das strukturierte Ableiten von Testdaten durch Äquivalenzklassenbildung oder die Nutzung von Methoden zur Anweisungs-, Zweig- und Pfad-Überdeckung. Agile Entwicklungsmethoden verfolgen oft einen Test-first-Ansatz, bei dem nach der fachlichen Analyse direkt Tests erstellt werden und erst danach die konkrete Implementierung erfolgt. Des Weiteren wird das Modell-basierte Testen verwendet, bei dem Testfälle teilweise oder komplett aus Modellen abgeleitet werden [10], [11],[8].

9.8.4 Wiederverwendung und Konfigurabilität

Die Wiederverwendung von bestehenden Systemen oder Komponenten im Entwurfsprozess hat die Informatik mit vielen anderen technischen Disziplinen gemein. Dort betrifft die Wiederverwendung dann eher Hardware oder reale Bauteile vom Kleinen bis ins Große. Auch wiederverwendete Softwareeinheiten können von ganzen Anwendungssystemen über Komponenten bis hin zu einzelnen Objekten oder Funktionen reichen.

Die Wiederverwendung von Objekten oder Funktionen ist bereits gängige Praxis, wie z. B. durch die Verwendung von Standardbibliotheken. Für die Wiederverwendung von einzelnen Softwarekomponenten (von Subsystemen bis einzelnen Objekten) ist es notwendig, diese unabhängig vom Gesamtsystem zu entwickeln, über Schnittstellen konfigurierbar zu machen, Standardschnittstellen zu benutzen. Man benötigt eine Middleware, die die Kommunikation zwischen den Komponenten steuert und man muss die Entwicklungsprozesse für die komponentenbasierte Entwicklung anpassen. Die Wiederverwendung von Komponenten kann durch Entwurfsentscheidungen erschwert werden. Zu einem guten Programmierstil gehört daher auch die Verwendung von Entwurfsmustern [12].
Die Wiederverwendung von Anwendungssystemen kann durch Konfiguration z. B. durch Verwendung domänenspezifischer Sprachen (DSLs) [13] oder die Entwicklung von Produktlinien für verschiedene Kunden angepasst werden oder man kommt durch die Zusammenfassung mehrerer Systeme zu einer neuen Anwendung. Zu letzterem wurde bereits in [14] angemerkt, dass die Kopplung von Werkzeugen über wohldefinierte Schnittstellen hierfür eine sinnvolle und anzustrebende Maßnahme für bestehende Lösungen wäre. Betrachten wir nun zwei Varianten zur besseren Konfigurierung von Systemen etwas genauer.

9.8.4.1 Variabilität und Konfigurabilität durch Produktlinien

Der Einsatz von Software in realen Umgebungen macht es oft notwendig, unterschiedliche Varianten eines Produkts auf z. B. derselben Hardware, Sensoren, Aktuatoren für unterschiedliche Arten von Gebäuden in unterschiedlichen Versionen auszuliefern bzw. diese auch für unterschiedliche Varianten der Hardware variabel veränderbar zu gestalten. Hierdurch wird es einerseits notwendig, Informationen darüber zu haben, welche Produktstände bei welchen Kunden in Betrieb sind, um insbesondere bei Updates oder Bugfixes die entsprechenden Systeme bzw. Altsysteme (legacy) zu identifizieren und aktualisieren und andererseits die Entwicklungsprozesse von Anfang an dahingehend zu optimieren, Softwarekomponenten leicht wiederverwendbar bzw. anpassbar zu gestalten.
Im Software Engineering wurde für das Variantenmanagement das Konzept der Produktlinien eingeführt, dass man bereits gut im produzierenden Gewerbe wie der Automobilbranche kennt. Softwareproduktlinien (SPL) sind eine Sammlung an Anwendungen, die eine gemeinsame Problembereichspezifische Architektur besitzen [2]. Ein gemeinsamer Anwendungskern wird jedes Mal weiterverwendet, einige Komponenten zur Bereitstellungs- oder zur Entwurfszeit konfiguriert, zusätzliche implementiert oder an neue Anforderungen angepasst.
Für die SPL Entwicklung [15] sind jedoch zwei parallele Stränge von Entwicklungsprozessen notwendig (das abstrahierte Domänen-Engineering für die allgemeinen, wiederverwendbaren Teile und das Applikations-Engineering für eine konkrete Anwendung), man benötigt einen guten Überblick über die realisierten Varianten, z. B. durch die Nutzung von Feature Diagrammen und muss insbesondere für die ersten beiden realisierten Systeme mit höheren Entwicklungszeiten rechnen, bis man dann langsam den Break-Even erreicht. Softwareproduktlinien machen es dann jedoch möglich, eine Vielzahl von Varianten bei besser planbaren Kosten in einer kürzeren Entwicklungszeit realisieren zu können.

9.8.4.2 Konfigurabilität durch DSLs

Eine andere Möglichkeit, Komponenten und Systeme gut wiederverwenden zu können, ist die Konfigurierung durch domänenspezifische Sprachen (DSLs). Mit Hilfe eigens definierter DSLs kann beispielsweise die strukturierte Planung von Performance-Zielen von Gebäuden bis hin zum konkreten Regelungsverhalten zur Laufzeit unterstützt werden [16]. zeigt hier bspw. einen Ansatz zur Optimierung der Energieeffizienz von Gebäuden und technischen Anlagen, in dem Regeln dazu verwendet werden, das funktionale Verhalten von Komponenten zu beschreiben und mit deren tatsächlichem Verhalten während des Betriebs abzugleichen. Beispiele für die Nutzung von DSLs zur Entwicklungszeit sind z. B. Architekturmodelle, um mögliche Variationspunkte von Komponenten zu beschreiben oder Delta-Modelle, um die Unterschiede zwischen Varianten abzubilden [5].

9.8.5 Zusammenfassung

Die Forschung zu energieoptimierten Gebäuden und Quartieren besteht in vielen Fällen aus einem Zusammenspiel von Software und Hardware. Durch den Einsatz von Methoden des Software Engineering wie beispielsweise adäquate Softwarearchitekturen, die generative Entwicklung von Benutzeroberflächen sowie die Testautomatisierung und wiederverwendbare Gestaltung von Objekten, Komponenten und Anwendungssystemen kann die Softwarequalität nachhaltig verbessert werden und schafft die Basis für langlebige, wartbare und wiederverwendbare Systeme.

Literatur:

[1] Greifenberg, T., Look, M., Pinkernell, C., Rumpe, B.: Energieeffiziente Städte–Herausforderungen und Lösungen aus Sicht des Software Engineerings. In: Linnhoff-Popien C., Zaddach M., Grahl A. (Hrsg.), Marktplätze im Umbruch: Digitale Strategien für Services im Mobilen Internet, Springer Berlin Heidelberg, 2015.

[2] Sommerville, I., Software Engineering, 10. Aufl., Pearson Studium, 2018.

[3] Newman, S., Building Microservices, 1. Aufl., O'Reilly, 2015.

[4] Martin, R. C., Clean Architecture, 1. Aufl., Prentice Hall, 2017.

[5] Hölldobler, K., Michael, J., Ringert, J. O., Rumpe, B., Wortmann, A., Innovations in Model-based Software and Systems Engineering, The Journal of Object Technology, 18, 2019, 1-60.

[6] Adam, K., Michael, J., Netz, L., Rumpe, B., Varga, S., Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: Digital Ecosystems of the Future: Methods, Techniques and Applications (EMISA'19), LNI, 2019.

[7] Michael, J., Steinberger, C. Shekhovtsov, V. A., Al Machot, F., Ranasinghe, S., Morak, G., The HBMS Story - Past and Future of an Active Assistance Approach, Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling, 13, 2018, 345-370.

[8] Vigenschow, U., Testen von Software und Embedded Systems, 2. Aufl., dpunkt.verlag, 2010.

[9] Myers G.J., Sandler C., Badgett T., The Art of Software Testing, 3. Aufl., Wiley, 2011.

[10] Rumpe B., Agile Modeling with UML: Code Generation, Testing, Refactoring, Springer International, 2017.

[11] Drave, I., Greifenberg, T., Hillemacher, S. Kriebel, S., Markthaler, M. Rumpe, B., Wortmann, A., Model-Based Testing of Software-Based System Functions., in: Conf. on Software Engineering and Advanced Applications (SEAA'18), Prague, 2018.

[12] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns. Elements of Reusable Object-Oriented Software, Prentice Hall, 1994.

[13] Clark, T., v. d. Brand, M., Combemale, B., Rumpe, B., Conceptual Model of the Globalization for Domain-Specific Languages. In: Globalizing Domain-Specific Languages (Dagstuhl Seminar), LNCS 9400, Springer, 2015, 7-20.

[14] Kirnats, L., Joost, J.-N., Berg, S., Frisch, J., van Treeck, C., Status Quo bei digitalen Werkzeugen und softwarebasierten Lösungsansätzen. Bauphysik, 40, 2018, 441-

448.

[15] Pohl K., Metzger A., Software Product Lines, in: Gruhn V., Striemer R. (eds) The Essence of Software Engineering. Springer, Cham, 2018.

[16] Kurpick, T., Look, M., Pinkernell, C., Rumpe B., Modeling Cyber-Physical Systems: Model-Driven Specification of Energy Efficient Buildings, in: Proc. of the Modelling of the Physical World Workshop MOTPW ,12, ACM Digital Library, 2012.

Kontakt:

RWTH Aachen
Software Engineering

Jörg Christian Kirchhof, M.Sc.
kirchhof@se-rwth.de

Dr. Judith Michael
michael@se-rwth.de

Prof. Dr. Bernhard Rumpe
rumpe@se-rwth.de