Vorwort

Und jedem Anfang wohnt ein Zauber inne ...
Hermann Hesse

Warum ein Buch zum Thema Modularisierung von Software? Ist dazu nicht bereits alles geschrieben worden? Ist das nicht längst alles klar? Das klappt doch praktisch in allen Softwarehäusern, oder etwa nicht? Was ist also die Motivation eines Autors, gerade darüber ein Buch zu schreiben, und das im 21. Jahrhundert? Um es auf den Punkt zu bringen: Ich glaube keineswegs, dass dieses Thema nirgends mehr Herausforderungen bietet. Ich denke, dass der Grund dafür, dass viele nach wie vor nach Patentrezepten für Softwarearchitektur suchen, genau der ist, dass sie sich fragen, wie man denn eine komplexe Software effizient in ihre Einzelteile zerlegen kann. Der aktuelle Microservice-Hype ist da nur ein Beispiel dafür. Der Wunsch nach einem solchen Allheilmittel für die Softwarearchitektur birgt allerdings ein paar Risiken in sich:

- Man redet aneinander vorbei. Viele haben andere Vorstellungen von den jeweiligen Architekturstilen. Gerade bei SOA und Microservices sind Missverständnisse an der Tagesordnung.
- Das Patentrezept wird angewendet, ohne Rücksicht auf Verluste. Es ist dann kein Mittel zum Zweck mehr, sondern das eigentliche Ziel der Architekturarbeit. Es wird irgendwann gar nicht mehr hinterfragt, ob das Patentrezept für den jeweiligen Anwendungsfall überhaupt sinnvoll ist.

Nach der Lektüre dieses Buchs sehen Sie, lieber Leser, hoffentlich gar nicht mehr den Bedarf nach Patentrezepten, um Ihre Software-Lösung zu strukturieren. Diese sind nämlich gar nicht notwendig, wenn man erstmal die folgenden Dinge verinnerlicht hat:

- Warum man strukturiert, wann man dies besser unterlässt und welche Vor-, aber auch Nachteile die Modularisierung von Software haben kann.
- Welche Möglichkeiten zur Modularisierung bestehen, wann man welche davon anwendet und welche Vor- und Nachteile diese haben.
- Wie man die geeigneten Grenzen für die Modularisierung identifiziert.

Über SOA und Microservices

Die Älteren unter meinen Lesern erinnern sich vielleicht noch an den SOA-Hype!? Jahrelang galt es als hip, genau das zu tun, wenn man eine sehr komplexe Software im Unternehmenskontext entwickelt hatte. Es dauerte eine Weile, bis sich zeigte, dass, aus meiner Sicht und der Meinung mehr oder weniger aller Softwarearchitekten zu dem Thema heute, diese Art von Patentrezept nicht funktioniert. SOA-Projekte scheiterten, was aber nicht zwangsläufig dazu führte, dass man diese auch vorzeitig beendete. Oft sind Organisationen hier nicht in der Lage, sich ein Scheitern einzugestehen. Man hält an dem einmal eingeschlagenen Weg so lange fest, bis sich der Fehlschlag nicht mehr leugnen lässt, weil die Organisation sonst ihr Gesicht verlieren könnte. Was wir heute als Überbleibsel dieses Hypes nach wie vor haben, sind Enterprise-Architekturen voller technischer Schuld, inklusive der deswegen unzufriedenen Auftraggeber und frustrierten Entwickler. Lösungsrezepte als Auswege aus solch verfahrenen Situationen liefert dieses Buch in Kapitel 9.

Aktuell wurde der SOA-Hype vom Microservice-Hype beinahe 1:1 abgelöst, wobei viele Enterprise-Architekten – nicht ganz zu Unrecht – daran zweifeln, ob dieser Architekturstil auch für eine komplexe Unternehmensanwendung brauchbar ist. Viele übersehen dabei, dass, ganz ähnlich wie beim SOA-Hype, nicht klar ist, wie genau denn dieser Architekturstil definiert ist. Ich sehe aktuell verschiedene mögliche Interpretationen, die sich zwar in gewissen Grundzügen ähneln, aber doch auf verschiedene Ziele hin optimiert sind. Ich werde auf diese im weiteren Verlauf des Buchs natürlich eingehen und auch erklären, warum der SOA-Hype gescheitert ist.

Womit wir bei einem leidigen Thema wären, welches oft Debatten im Bereich der Softwarearchitektur schwierig macht, nämlich der Art und Weise, wie wir Fachbegriffe benutzen. Anders als Sparten wie Medizin oder Biologie haben wir es bisher versäumt, eine allgemein anerkannte Begriffswelt zu definieren. Inzwischen gibt es einen beachtenswerten Versuch des iSAQB, wo es allerdings Erfolg und Verbreitungsgrad noch abzuwarten gilt. Solange wir eine solche gemeinsame Sprache nicht haben, bleibt auch mir als Autor nichts anderes übrig, als die Begriffe, die ich in diesem Buch verwende, explizit zu definieren. Dem Thema habe ich daher am Ende dieses Buchs einen Abschnitt gewidmet.

Apropos Anhang des Buchs. Da viele der Muster und Prinzipien, die wir in diesem Buch besprechen, in mehr als einem Abschnitt diskutiert werden, habe ich diese mit eindeutigen Identifiern versehen, über die ich mich immer wieder darauf beziehe, wie P02-SoC für das Prinzip „Separation of Concerns“. Diese sind im Anhang ebenfalls nochmal zusammengefasst und außerdem in Relation zueinander gestellt. Dasselbe gilt für diverse Antipattern (A??-) und Gesetze wie Conway’s Law (L??-), die dort ebenfalls angeführt sind. Darüber hinaus finden Sie dort noch klassische Trade-offs (T??-) der Softwarearchitektur. Dabei handelt es sich um Zwickmühlen, aus denen es keinen klaren Ausweg gibt. Im konkreten Fall muss man jeweils abwägen, für welche der Optionen man sich entscheidet.

Was ich aber zuerst klären möchte, u. a. weil eine klare und allgemeine Definition fehlt, ist, was ich unter dem Begriff Softwarearchitektur verstehe und worum es genau in diesem Buch eigentlich gehen wird. Diesem Thema widme ich gleich im Anschluss das erste Kapitel.

Als roter Faden durch dieses Werk wird sich ein Tutorial zum Thema Modularisierung von Code ziehen. Die Beispiele dafür (genannt „Labs“) finden Sie auf der Website des Verlags für Java (*https://downloads.hanser.de/*). Es gibt außerdem noch eine etwas veraltete Version

davon in C# auf github unter *https://github.com/hdowalil/software-design-c_sharp* zu finden. Diese basiert noch auf .NET Core 2.0, was aber auf den Code selbst keinen Einfluss hat. Der C#-Code wird von mir nicht mehr gewartet, ist aber, wenn Sie C# der Sprache Java deutlich bevorzugen, durchaus brauchbar.

Um die Labs auf Ihrem Rechner nachvollziehen zu können, benötigen Sie ein JDK (mindestens Version 9) und eine IDE, die Java-Projekte kompilieren kann, die auf Maven basieren. Sehr gut klappt das mit der „Eclipse IDE for Java Developers“, in der das m2e-Plugin für die Maven-Integration in Eclipse bereits inkludiert ist. Einfach das .zip-File lokal entpacken und in Eclipse beide Projekte (training-clean und training-design) mit „Import Project“ und danach „Existing Maven Project“ importieren.

Der Aufbau der einzelnen Labs ist dabei immer derselbe. Es gibt jeweils ein Negativ- („Challenge“) und ein Positiv-Beispiel („Solution“), welche auch in diesem Buch diskutiert werden („Sample“). Dazu noch eine weitere Übung („Excercise“), an der Sie sich selbst versuchen können, um im Endeffekt Ihre Lösung („Solution/Your“) mit der Vorgabe des Autors („Solution/Trainer“) zu vergleichen. Dabei wünsche ich Ihnen viel Spaß!

Die Notation der Abbildungen

Mir sind die Vorteile eines Standards wie der Unified Modelling Language (UML) natürlich vollkommen bewusst. Ich habe auch für die 2. Auflage meines Buchs beschlossen, explizit (mit wenigen Ausnahmen) nicht auf diese zu setzen. Meiner Erfahrung nach bringt die enorme Komplexität der UML auch gewisse Gefahren mit sich. Wussten Sie beispielsweise, dass unterschiedliche Pfeilspitzen-Darstellungen im Sequenzdiagramm bedeuten, dass die jeweilige Kommunikation synchron oder asynchron passiert? In der Theorie bietet einem die UML den Vorteil, dass sie ungemein aussagekräftig ist, allerdings nur, wenn jedem, der die Diagramme bearbeitet und liest, auch all diese Feinheiten bewusst sind. Wenn also jemand synchrone und asynchrone Kommunikation im Diagramm festhält, sollte der Leser, denselben Wissensstand vorausgesetzt, genauestens Bescheid wissen. In der Praxis erlebe ich aber oft, dass es genau deswegen zu Missverständnissen kommt. Entweder wusste der Autor des Diagramms nichts darüber und das Diagramm wird vom Leser fehlinterpretiert, oder aber der Leser übersieht diese möglicherweise wichtige Information. Der Nachteil, wenn man nicht auf einen Standard wie die UML setzt, ist der, dass man die eigene Notation erklären muss. Ich mache das vor allem, was die Bedeutung der Pfeilrichtungen angeht: Es geht in meinen Diagrammen, wenn diese nicht auf der UML basieren, meist um statische Abhängigkeit, nicht etwa um zeitliche Reihenfolge oder Datenflüsse. Nur wenn die Pfeile auch mit Zahlen notiert sind, geht es um den Ablauf einer Interaktionssequenz.

Danksagung

Danken möchte ich allen, die mich zu diesem Buch inspiriert und ermutigt haben, sowie meiner Frau und meiner Tochter, die mir den Freiraum dafür geschaffen und es mir ermöglicht haben. Inspiriert zu den Inhalten wurde ich von aktuellen und ehemaligen Kollegen wie Stefan Toth, Stefan Zörner, Peter Götz, Rene Weiss, Oliver Zeigemann, Mohammad

Kabiri, Michael Koitz, Miroslaw Szelag, Christa Dihanich, Jürgen Kofler, Milan Heimschild und Sebastian Bicchi. Ermutigt, meine Inhalte als Buch zu veröffentlichen, haben mich in erster Linie Gernot Starke und Alexander von Zitzewitz. Danken möchte ich aber auch jenen, die mit einer entweder hoffnungslos veralteten Einstellung oder einer bizarren Überzeugung zum Thema Softwarearchitektur meinen Skill zur Argumentation zu diesem Thema geschärft haben. Letztere werde ich an dieser Stelle allerdings nicht namentlich anführen.