Clean Architecture
Praxisbuch
für saubere Software-Architektur und wartbaren Code

DAS VORWORT

Vorwort

Das Paradies fürs Entwicklungsteam: Domänenlogik ist einfach zu testen, Infrastruktur und Technologie für Tests einfach zu simulieren (zu »mocken«) und es gibt eine ganz saubere Trennung zwischen Domänencode und technischem Code. Selbst die Migration von einer Technologie auf eine andere geht leicht von der Hand. Keine endlosen Diskussionen mehr, welcher Teil des Codes dieses verzwickte kleine Feature implementieren soll, das die Verkaufsleute schon der Fachabteilung versprochen haben. Dieses Paradies heißt »Clean Architecture«, und Tom wird Sie auf Ihrer Reise dorthin begleiten.

Seit einigen Jahren gibt es diese Clean Architecture unter verschiedenen Namen (etwa: Hexagonale Architektur, Ports-und-Adapter-Architektur, Zwiebelarchitektur). Dem Ganzen liegt eine ziemlich einfache Idee zugrunde: zwei konzentrische Kreise, die Domäne und Technik innerhalb der Software trennen. Abhängigkeiten fließen grundsätzlich nach innen, von der Technologie zur Domäne. Domänenklassen dürfen niemals Abhängigkeiten zu technischen Klassen haben.

Zu schade nur, dass die meisten der Originalquellen es versäumt haben, die Details der Umsetzung zu erklären, beispielsweise wie Packages und Code organisiert sein sollten. Toms Buch füllt diese Lücke perfekt aus. Anhand eines anschaulichen Beispiels führt Tom Sie (und Ihr Entwicklungsteam) zu einer ausgesprochen gut wartbaren und sauberen architektonischen Struktur.

Tun Sie sich selbst und Ihrem Entwicklungsteam einen Gefallen, geben Sie der Clean Architecture eine Chance. Ich verspreche Ihnen, dass Sie es nicht bereuen werden!

Gernot Starke
Köln, Juni 2023
Pragmatischer Softwarearchitekt seit den 1990er-Jahren, Gründer von arc42, Mitgründer von iSAQB und Nerd

Einleitung

Wenn Sie dieses Buch in die Hand genommen haben, dann machen Sie sich vermutlich Gedanken um die Software, die Sie entwickeln. Sie möchten nicht nur, dass Ihre Software die expliziten Anforderungen Ihrer Kunden erfüllt, sondern auch die implizite Anforderung der Wartbarkeit sowie Ihre eigenen Ansprüche hinsichtlich Struktur und Ästhetik.

Es ist schwer, diesen Anforderungen gerecht zu werden, da Softwareprojekte (oder Projekte ganz allgemein, wenn wir ehrlich sind) üblicherweise nicht so verlaufen wie geplant. Manager ziehen um das ganze Projektteam eine Deadline[1], externe Partner bauen ihre APIs anders als versprochen und die Softwareprodukte, von denen wir abhängig sind, funktionieren nicht so, wie wir es erwarten.

Und dann ist da noch Ihre eigene Softwarearchitektur. Zu Anfang war sie so einfach. Alles war klar und schön. Dann zwang die Deadline Sie dazu, Abkürzungen zu nehmen. Die Abkürzungen sind nun alles, was von der Architektur geblieben ist, und es dauert länger und länger, neue Features abzuliefern.

Ihre von Abkürzungen belastete Architektur macht es schwer, auf eine API zu reagieren, die geändert werden musste, weil ein externer Partner Mist gebaut hat. Es scheint leichter, einfach Ihren Projektmanager vorzuschicken, um diesem Partner mitzuteilen, dass er die API abliefern soll, auf die Sie sich geeinigt hatten.

Inzwischen haben Sie vollständig die Kontrolle über die Situation verloren. Mit hoher Wahrscheinlichkeit wird eines der folgenden Dinge passieren:

- Der Projektmanager ist nicht stark genug, um den Kampf gegen den externen Partner zu gewinnen.
- Der externe Partner findet ein Schlupfloch in der Spezifikation der API, das ihm recht gibt.
- Der externe Partner braucht weitere <hier Zahl einsetzen> Monate, um die API anzupassen.

1 Das Wort »Deadline« stammt angeblich aus dem 19. Jahrhundert und beschrieb eine Linie, die um ein Gefängnis oder Gefangenenlager gezogen wurde. Ein Gefangener, der diese Linie überquerte, wurde erschossen. Behalten Sie das im Hinterkopf, wenn wieder einmal jemand eine »Deadline« zieht! Im Deutschen gibt es den viel harmloseren – wenn auch bürokratischer klingenden – Begriff des »Fälligkeitstermins«. Falls Sie gern gefährlicher leben, können Sie auch von einer »Galgenfrist« sprechen – das ist die kurze Zeit, die einem noch bis zu einem unangenehmen Ereignis (dem Gang zum Galgen) bleibt.

All das führt zum selben Ergebnis ? Sie müssen schnell Ihren Code ändern, weil die Deadline droht.

Sie fügen eine weitere Abkürzung ein.

Anstatt den Zustand Ihrer Softwarearchitektur durch äußere Umstände diktieren zu lassen, setzt dieses Buch darauf, dass Sie selbst die Kontrolle übernehmen. Sie erreichen dies, indem Sie eine Architektur erschaffen, die Ihre Software »soft«, also »weich« macht, im Sinne von »flexibel«, »erweiterbar« und »anpassbar«. Eine solche Architektur erlaubt es Ihnen problemlos, auf externe Faktoren zu reagieren und nimmt eine große Last von Ihren Schultern.

Das Ziel dieses Buches

Ich habe dieses Buch geschrieben, weil ich von der Praktikabilität der verfügbaren Ressourcen über domänenzentrierte Architekturstile enttäuscht war. Zu diesen Architekturstilen gehören *Clean Architecture* von Robert C. Martin und *Hexagonal Architecture* von Alistair Cockburn.

Viele Bücher und Online-Ressourcen beschreiben wertvolle Konzepte, aber nicht, wie man diese tatsächlich implementieren kann. Das liegt vermutlich daran, dass es mehr als eine Möglichkeit gibt, einen Architekturstil zu implementieren.

Mit einer praktischen Diskussion über das Herstellen einer Webanwendung im Stil einer Hexagonalen Architektur bzw. im »Ports-und-Adapter«-Stil versuche ich, diese Lücke zu füllen. Um dieses Ziel zu erreichen, demonstrieren die Codebeispiele und Konzepte in diesem Buch meine Interpretation der Implementierung einer Hexagonalen Architektur. Es gibt ganz sicher andere Interpretationen und ich erhebe auch nicht den Anspruch, dass meine *die* allgemeingültige Lösung ist.

Ich hoffe jedoch, dass die Konzepte in diesem Buch Ihnen eine gewisse Inspiration bieten, sodass Sie Ihre eigene Interpretation der Hexagonalen/Clean Architecture finden.

An wen sich dieses Buch richtet

Dieses Buch ist für Softwareentwickler aller Erfahrungsstufen gedacht, die sich mit dem Erstellen von Webanwendungen befassen.

Als Einsteiger oder Einsteigerin lernen Sie, wie Sie Softwarekomponenten und ganze Anwendungen auf saubere und wartbare Art und Weise entwerfen. Sie erfahren auch etwas darüber, wann Sie eine bestimmte Technik einsetzen können und sollten. Um wirklich den größten Nutzen aus diesem Buch zu ziehen, sollten Sie allerdings bereits einmal an der Erstellung einer Webanwendung mitgewirkt haben.

Als Entwicklerin und Entwickler mit größerer Erfahrung haben Sie Gelegenheit, die Konzepte aus dem Buch mit Ihrer eigenen Vorgehensweise zu vergleichen und hoffentlich einige davon in Ihren eigenen Entwicklungsstil zu integrieren.

Die Codebeispiele sind in Java und Kotlin geschrieben, aber alle Erörterungen lassen sich gleichermaßen auf andere objektorientierte Programmiersprachen anwenden. Falls Sie kein Java-Programmierer sind, aber objektorientierten Code in anderen Sprachen lesen können, ist alles gut. An den wenigen Stellen, an denen wir Java- oder Framework-Spezifika benötigen, werde ich diese erklären.

Die Beispielanwendung

Um einen thematischen roten Faden in diesem Buch zu haben, zeigen die meisten Codebeispiele Ausschnitte einer beispielhaften Webanwendung für Online-Überweisungen. Sie trägt den Namen »BuckPal«[2].

Die BuckPal-Anwendung erlaubt es einem Benutzer, ein Konto zu registrieren, Geld zwischen Konten zu transferieren und die Aktivitäten auf dem Konto (Ein- und Auszahlungen) einzusehen.

Ich bin kein Finanzspezialist, also versuchen Sie nicht, den Beispielcode anhand seiner rechtlichen oder funktionalen Korrektheit zu bewerten. Bewerten Sie ihn lieber anhand seiner Struktur und seiner Wartbarkeit.

Der Fluch von Beispielanwendungen für Softwaretechnikbücher und Online-Ressourcen liegt darin, dass sie zu einfach sind, um tatsächlich die realen Probleme anzusprechen, mit denen Sie sich jeden Tag auseinandersetzen müssen. Andererseits muss eine Beispielanwendung einfach genug bleiben, um die diskutierten Konzepte effektiv zu vermitteln.

Ich hoffe, das richtige Gleichgewicht zwischen »zu einfach« und »zu komplex« gefunden zu haben, wenn wir die Anwendungsfälle der BuckPal-Anwendung in diesem Buch diskutieren.

Der Code der Beispielanwendung ist auf GitHub zu finden.[3]

2 Eine schnelle Online-Suche hat ergeben, dass ein Unternehmen namens PayPal meine Idee gestohlen und sogar einen Teil des Namens kopiert hat. Scherz beiseite: Versuchen Sie einmal, einen Namen zu finden, der so ähnlich ist wie »PayPal«, aber nicht der Name eines existierenden Unternehmens ist. Zum Schreien komisch!

3 Das BuckPal-GitHub-Repository: `https://github.com/thombergs/buckpal`.

Treten Sie in Kontakt

Falls Sie etwas zu diesem Buch sagen möchten, würde ich mich freuen, von Ihnen zu hören. Schreiben Sie mir direkt eine E-Mail an `tom@reflectoring.io` oder auf Twitter über `@TomHombergs`.

Sollte es zu diesem Buch bereits eine Errata-Liste geben, ist sie unter `www.mitp.de/0814` zu finden.