O'REILLY®

US-Bestseller
Übersetzung der 2. Auflage

Prinzipien des Software-designs

Entwurfsstrategien für komplexe Systeme

John Ousterhout
Übersetzung von Thomas Demmig

Inhalt

KAPITEL 3

Funktionierender Code reicht nicht aus (strategische versus taktische Programmierung)

Eines der wichtigsten Elemente guten Softwaredesigns ist die mentale Haltung bei einer Programmieraufgabe. Viele Organisationen unterstützen eine taktische Herangehensweise und konzentrieren sich darauf, Features so schnell wie möglich zum Laufen zu bringen. Aber wenn Sie ein gutes Design haben wollen, müssen Sie eher einen strategischen Ansatz verfolgen, bei dem Sie Zeit investieren, um saubere Designs zu schaffen und Probleme zu beheben. Dieses Kapitel dreht sich darum, warum der strategische Ansatz für bessere Designs sorgt und langfristig tatsächlich günstiger ist als ein taktisches Vorgehen.

Taktische Programmierung

Die meisten Menschen gehen die Softwareentwicklung mit einer mentalen Haltung an, die ich als *taktische Programmierung* bezeichne. Dabei geht es ihnen vor allem darum, etwas zum Laufen zu bringen, wie zum Beispiel ein neues Feature oder einen Bugfix. Auf den ersten Blick ist das total vernünftig: Was könnte wichtiger sein, als Code zu schreiben, der funktioniert? Trotzdem sorgt taktische Programmierung dafür, dass ein gutes Systemdesign nahezu unmöglich wird.

Das Problem bei der taktischen Programmierung ist ihre Kurzsichtigkeit. Programmieren Sie taktisch, versuchen Sie, eine Aufgabe so schnell wie möglich fertigzustellen. Vielleicht haben Sie eine harte Deadline. Dann hat zukunftsorientiertes Planen keine so hohe Priorität. Sie wenden nicht viel Zeit auf, nach dem besten Design Ausschau zu halten – Sie wollen nur, dass etwas schnell funktioniert. Sich selbst beruhigen Sie damit, dass es in Ordnung ist, ein bisschen Komplexität ins System zu bringen oder ein oder zwei kleine schmutzige Tricks anzuwenden, wenn die aktuelle Aufgabe damit schneller abgeschlossen ist.

So werden Systeme komplizierter. Wie im vorherigen Kapitel besprochen, ist Komplexität inkrementell. Es ist nicht diese eine besondere Sache, die ein System komplizierter macht, sondern das Aufsummieren Dutzender oder Hunderter kleiner Dinge. Programmieren Sie taktisch, wird jede

Programmieraufgabe ein bisschen zu dieser Komplexität beitragen. Jeder Schritt scheint ein vernünftiger Kompromiss zu sein, um die aktuelle Aufgabe schnell abzuschließen. Aber die Komplexität wächst schnell, insbesondere wenn alle taktisch programmieren.

Über kurz oder lang werden einige der Komplexitäten für Probleme sorgen, und Sie beginnen, sich zu wünschen, sie hätten frühere Workarounds nicht genutzt. Aber Sie werden sich einreden, dass es wichtiger ist, das nächste Feature fertigzustellen, als einen Schritt zurückzugehen und den bestehenden Code zu refaktorieren. Refaktorieren mag Ihnen langfristig helfen, aber es verlangsamt definitiv die aktuelle Aufgabe. Also suchen Sie nach schnellen Korrekturen, um Probleme zu umgehen, denen Sie sich gegenübersehen. Das sorgt für noch mehr Komplexität, was weitere schnelle, schmutzige Fixes erfordert. Ziemlich schnell ist der Code ein großes Durcheinander, aber jetzt ist alles schon so verworren, dass es Monate brauchen würde, das Ganze aufzuräumen. Solch eine Verzögerung würde Ihr Zeitplan nicht zulassen, und das Beheben von ein oder zwei Problemen scheint keinen großen Unterschied auszumachen, also bleiben Sie bei der taktischen Programmierung.

Sofern Sie sehr lange an einem großen Softwareprojekt gearbeitet haben, ist Ihnen taktische Programmierung vermutlich bereits begegnet, und Sie sind auch schon über die Probleme gestolpert, die diese verursacht. Haben Sie einmal den taktischen Weg eingeschlagen, ist es schwer, die Richtung zu ändern.

In so gut wie jeder Softwareentwicklungsorganisation gibt es mindestens eine Person, die taktische Programmierung ins Extrem treibt: ein *taktischer Tornado*. Diese Person ist sehr produktiv und wirft Code schneller als alle anderen aus, geht aber vollkommen taktisch vor. Geht es darum, mal eben ein Feature zu implementieren, ist keiner so schnell wie der taktische Tornado. In manchen Organisationen behandelt das Management die taktischen Tornados als zu verehrendes Vorbild. Aber sie hinterlassen eine Schneise der Verwüstung. Diejenigen, die in Zukunft mit ihrem Code arbeiten müssen, verehren sie eher selten. Meist müssen andere später das Chaos aufräumen, das die Tornados mit ihrem Code hinterlassen haben – was dann so aussieht, als wäre der »Reinigungstrupp« (die wahren Helden) langsamer als der Tornado.

Strategische Programmierung

Der erste Schritt hin zu einem guten Softwaredesign ist die Erkenntnis, dass **funktionierender Code** nicht ausreicht. Es ist nicht akzeptabel, unnötige

Komplexität in das System zu bringen, um Ihre aktuelle Aufgabe schneller abschließen zu können. Am wichtigsten ist die langfristige Struktur des Systems. In jedem System entsteht ein Großteil des Codes dadurch, dass die bestehende Codebasis erweitert wird. Daher ist es Ihre Aufgabe, beim Entwickeln diese zukünftigen Erweiterungen zu ermöglichen. Sie sollten also nicht den funktionierenden Code als wichtigstes Ziel ansehen, auch wenn Ihr Code natürlich funktionieren muss. Vor allem müssen Sie ein sehr gutes Design schaffen, das zudem noch funktioniert. Das ist *strategisches Programmieren*.

Für das strategische Programmieren braucht man eine Investitionsmentalität. Statt den schnellsten Weg zum Abschluss Ihres aktuellen Projekts zu wählen, müssen Sie Zeit investieren, um das Design des Systems zu verbessern. Diese Investitionen werden Sie kurzfristig ein wenig verlangsamen, aber langfristig werden Sie damit schneller sein, wie Abbildung 3-1 demonstriert.

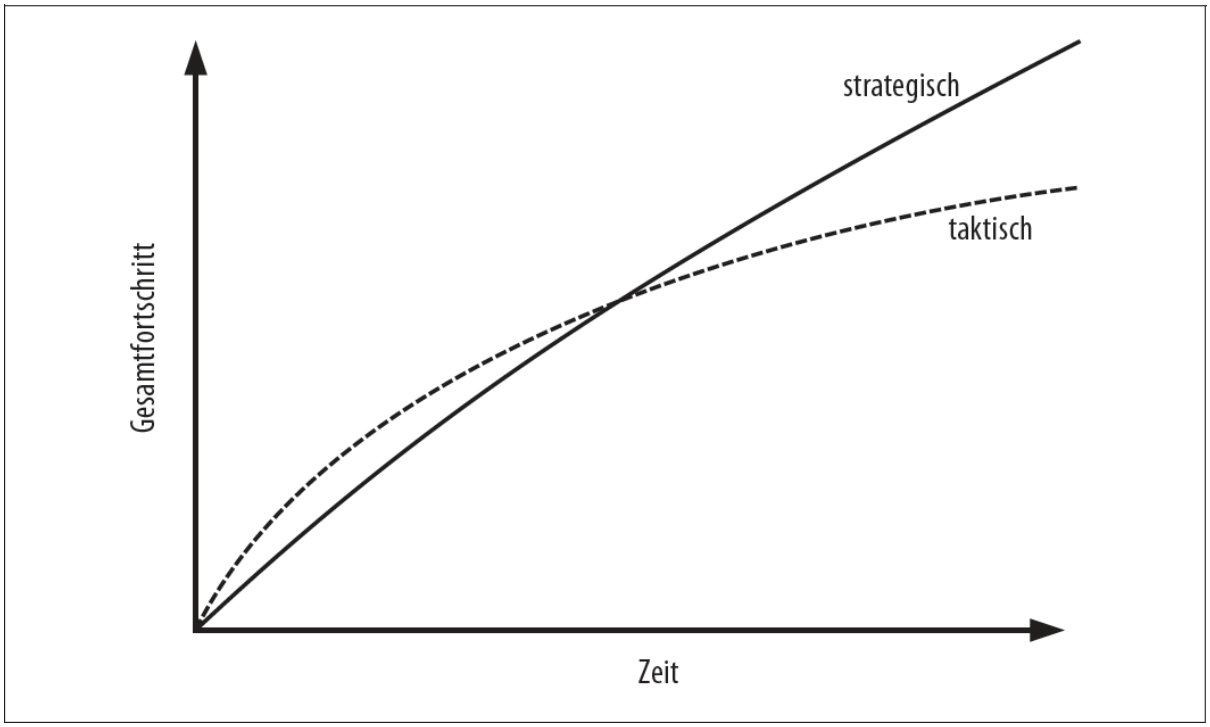

Abbildung 3-1: Zu Beginn wird ein taktisches Vorgehen beim Programmieren für einen schnelleren Fortschritt sorgen als ein strategischer Ansatz. Aber die Komplexität summiert sich beim taktischen Ansatz schneller auf, was die Produktivität verringert. Mit der Zeit führt das strategische Vorgehen zu schnellerem Fortschritt. Beachten Sie: Dieses Diagramm soll nur der qualitativen Veranschaulichung dienen – ich kenne keine empirischen Untersuchungen zum genauen Verlauf der Kurven.

Manche der Investitionen werden proaktiv sein. So lohnt es sich zum Beispiel, für jede neue Klasse ein wenig Zeit aufzuwenden, um ein einfaches Design zu finden – statt also die erste Idee zu implementieren, die Ihnen in den Sinn kommt, probieren Sie ein paar alternative Designs aus und wählen das sauberste davon. Versuchen Sie, sich ein paar Richtungen vorzustellen, in die sich das System in Zukunft ändern könnte, und stellen Sie sicher, dass diese mit Ihrem Design möglich sind. Das Schreiben guter Dokumentation ist ein weiteres Beispiel für eine proaktive Investition.

Andere Investitionen werden eher reaktiv sein. Egal, wie viel Sie im Voraus investieren – Sie werden bei Ihren Designentscheidungen immer Fehler machen. Irgendwann werden diese Fehler offensichtlich werden. Entdecken Sie ein Designproblem, ignorieren Sie es nicht einfach oder programmieren darum herum – nehmen Sie sich ein wenig Zeit, um es zu beheben. Wenn Sie strategisch programmieren, werden Sie immer wieder kleine Verbesserungen am Systemdesign vornehmen. Das ist das Gegenteil von taktischer Programmierung, bei der Sie fortlaufend für ein wenig mehr Komplexität sorgen, die in der Zukunft für Probleme sorgen wird.

Wie viel soll ich investieren?

Welches Ausmaß an Investitionen in ein besseres Design ist nun angemessen? Es wäre nicht effektiv, zu Beginn riesig viel Aufwand zu treiben, um zum Beispiel direkt das gesamte System zu designen. Das entspräche der Wasserfallmethode, von der wir wissen, dass sie nicht funktioniert. Das ideale Design tendiert dazu, sich Schritt für Schritt zu entwickeln, während Sie Erfahrungen mit dem System sammeln. Daher ist es am besten, kontinuierlich viele kleine Investitionen zu tätigen. Ich schlage vor, etwa 10 bis 20 % Ihrer gesamten Entwicklungszeit für Investitionen aufzuwenden. Das ist so wenig, dass es Ihre Zeitpläne nicht signifikant beeinträchtigt, aber so viel, dass es mit der Zeit signifikante Vorteile bringt. Ihre ersten Projekte werden daher um 10 bis 20 % länger dauern als bei einem rein taktischen Vorgehen. Diese zusätzliche Zeit führt aber zu einem besseren Softwaredesign, und Sie werden innerhalb weniger Monate die Vorteile erkennen. Es wird nicht lange dauern, bis Sie mindestens 10 bis 20 % schneller entwickeln, als wenn Sie taktisch programmieren würden. Ab diesem Punkt werden sich Ihre Investitionen auszahlen: Die Vorteile Ihrer früheren Investitionen werden Ihnen so viel Zeit sparen, dass die Kosten zukünftiger Investitionen damit abgedeckt sind. Sie werden die Kosten der ersten Investitionen schnell wieder eingespielt haben. Abbildung 3-1 verdeutlicht das.

Programmieren Sie hingegen taktisch, werden Sie Ihre ersten Projekte um 10 bis 20 % schneller abschließen, aber mit der Zeit wird sich Ihre Entwicklung verlangsamen, wenn sich die Komplexität ansammelt. Es wird nicht lange dauern, bis Sie mindestens 10 bis 20 % langsamer programmieren. Schnell werden Sie die Zeit verbraucht haben, die Sie zu Beginn einsparen konnten, und für die restliche Lebensdauer des Systems werden Sie langsamer entwickeln, als wenn Sie den strategischen Ansatz verfolgt hätten. Haben Sie noch nie in einer wirklich schlimm heruntergekommenen Codebasis gearbeitet, sprechen Sie mal mit einer Person, die das schon getan hat – sie wird Ihnen bestätigen, dass schlechte Codequalität die Entwicklung um mindestens 20 % verlangsamt.

Der Begriff *technische Schulden* wird oft verwendet, um die Probleme zu beschreiben, die durch taktische Programmierung verursacht werden. Mithilfe taktischen Programmierens borgen Sie sich Zeit aus der Zukunft: Die Entwicklung wird jetzt schneller sein, aber später mehr Zeit benötigen. Wie bei finanziellen Schulden werden Sie mehr zurückzahlen müssen, als Sie sich geborgt haben. Aber anders als bei finanziellen Schulden werden technische Schulden nie vollständig zurückgezahlt: Sie werden immer und immer wieder zahlen müssen.

Abbildung 3-1 wirft eine wichtige Frage auf: Wo schneiden sich die strategische und die taktische Kurve? Mit anderen Worten: Wie lange dauert es, bis sich der strategische Ansatz auszahlt? Ich kenne leider keine Daten zu diesem Thema, und es wäre auch schwierig, ein kontrolliertes Experiment so durchzuführen, dass die Antwort überzeugen könnte. Meiner persönlichen Meinung nach liegt der Zeitpunkt im Bereich zwischen 6 und 18 Monaten. Er hängt stark vom Erinnerungsvermögen der Entwicklerinnen und Entwickler ab: Ist ein Codeabschnitt ein paar Monate alt, hat man das meiste von dem vergessen, was man sich beim Schreiben dabei gedacht hat. Die Entwicklung wird dadurch signifikant verlangsamt, wenn der Code komplex ist. Diese zusätzlichen Kosten gleichen schnell einen initialen Vorsprung gegenüber der strategischen Programmierung aus. Aber wie gesagt: Das ist nur meine Meinung, und ich habe keine Daten, die das bestätigen können.

Start-ups und Investitionen

In manchen Umgebungen gibt es starke Kräfte, die gegen den strategischen Ansatz arbeiten. So spüren beispielsweise Start-ups in ihrer Frühphase einen unglaublichen Druck, erste Releases so schnell wie möglich herauszubringen. In solchen Firmen scheinen selbst die 10 bis 20 % Investitionen nicht machbar zu

sein. Daher verfolgen viele Start-ups einen taktischen Ansatz und wenden nur wenig Zeit für das Design und noch weniger für das Aufräumen nach Problemen auf. Sie begründen das damit, dass sie – wenn sie erfolgreich sind – genug Geld haben würden, um zusätzliches Personal zum Beseitigen des Chaos einstellen zu können.

Arbeiten Sie in einer Firma, die in diese Richtung tendiert, sollten Sie sich im Klaren sein, dass eine Codebasis, die einmal zu Spaghetticode wurde, so gut wie nie wieder aufgeräumt werden kann. Sie werden vermutlich über die gesamte Lebenszeit des Produkts hohe Entwicklungskosten haben. Zudem macht sich gutes (oder schlechtes) Design sehr schnell bemerkbar, daher stehen die Chancen nicht schlecht, dass der taktische Ansatz selbst Ihr erstes Produkt-Release nicht sehr beschleunigen wird.

Zudem sollten Sie berücksichtigen, dass einer der wichtigsten Faktoren für den Erfolg einer Firma die Qualität der Entwicklung ist. Am besten verringern Sie die Entwicklungskosten, indem Sie hervorragende Entwicklerinnen und Entwickler einstellen – sie kosten nicht viel mehr als mittelmäßiges Personal, haben aber eine sehr viel höhere Produktivität. Allerdings ist den besten ihrer Zunft gutes Design wichtig. Ist Ihre Codebasis Schrott, wird sich das herumsprechen, und es wird für Sie schwieriger werden, gutes Personal zu finden. So wird die Entwicklung nur mittelmäßig sein, was Ihre Kosten in Zukunft erhöhen und die Systemstruktur vermutlich noch schlechter machen wird.

Facebook ist ein Beispiel für ein Start-up, das taktische Programmierung gefördert hat. Viele Jahre lang hieß das Firmenmotto: »Move fast and break things.« Neues Personal frisch vom College wurde darin bestärkt, direkt in die Codebasis des Unternehmens einzutauchen – es war ganz normal, schon in der ersten Arbeitswoche Commits in die Produktivumgebung zu pushen. Das Positive war, dass Facebook sich den Ruf erwarb, eine Firma zu sein, die ihre Mitarbeitenden befähigt. Die Entwicklung hatte unglaublich viele Freiheiten, und es gab nur wenige Regeln und Einschränkungen.

Als Firma war Facebook spektakulär erfolgreich, aber die Codebasis hatte aufgrund des taktischen Ansatzes Schwächen – ein Großteil des Codes war instabil und nur schwer zu lesen, es gab wenige Kommentare oder Tests, und die Arbeit daran war schmerzvoll. Mit der Zeit erkannte das Unternehmen, dass seine Kultur nicht nachhaltig war. Schließlich änderte Facebook sein Motto zu: »Move fast with solid infrastructure.« Es unterstützte die Entwicklung darin, mehr in gutes Design zu investieren. Es wird sich zeigen, ob Facebook die

Probleme erfolgreich beseitigen kann, die sich in all den Jahren taktischer Programmierung angehäuft haben.

Fairerweise sollte ich darauf hinweisen, dass der Code von Facebook vermutlich nicht schlimmer ist als der eines durchschnittlichen Start-ups. Taktische Programmierung ist in vielen Start-ups verbreitet – Facebook ist dafür nur ein besonders sichtbares Beispiel.

Zum Glück ist es auch im Silicon Valley möglich, mit einem strategischen Ansatz erfolgreich zu sein. Google und VMware wurden ungefähr zur gleichen Zeit wie Facebook groß, aber beide Firmen verfolgten ein eher strategisches Vorgehen. Ihnen waren qualitativ hochwertiger Code und gutes Design wichtig, und beide Firmen schufen ausgefeilte Produkte, die komplexe Probleme mit zuverlässigen Softwaresystemen lösten. Die starke technische Kultur der Unternehmen wurde in Silicon Valley schnell bekannt. Nur wenige andere Firmen konnten mit ihnen beim Einstellen der besten Talente mithalten.

Diese Beispiele zeigen, dass Firmen sowohl mit dem einen als auch mit dem anderen Ansatz Erfolg haben können. Aber es macht viel mehr Spaß, in einer Firma zu arbeiten, der Softwaredesign wichtig ist und die eine saubere Codebasis besitzt.

Zusammenfassung

Gutes Design gibt es nicht umsonst. Sie müssen kontinuierlich investieren, sodass sich kleine Probleme nicht zu großen anhäufen. Glücklicherweise zahlt sich gutes Design aus – und zwar schneller, als Sie vielleicht denken.

Es ist entscheidend, beim Anwenden des strategischen Ansatzes konsequent vorzugehen und die Investitionen als etwas anzusehen, das Sie heute tun sollten und nicht erst morgen. Wird es zeitlich eng, ist es verlockend, das Aufräumen auf die Zeit nach einer Deadline zu verschieben. Aber das ist ein gefährlicher Weg – nach der aktuellen Deadline kommt mit Sicherheit die nächste und danach wieder eine. Haben Sie einmal damit begonnen, Verbesserungen nach hinten zu schieben, passiert es ganz schnell, dass Sie sie immer weiterschieben und sich Ihre Entwicklungskultur hin zum taktischen Ansatz bewegt. Je länger Sie damit warten, Designprobleme anzugehen, desto größer werden sie – die Lösungsansätze werden immer beängstigender, was dazu führt, dass Sie sie lieber noch mehr in die Zukunft schieben. Der effektivste Ansatz ist einer, bei dem jede Entwicklerin und jeder Entwickler fortlaufend kleine Investitionen in gutes Design tätigt.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z