SINN UND UNSINN VON FRAMEWORKS – TEIL II

Frameworks unterstützen und vereinfachen den Softwareentwicklungsprozess, indem sie zusätzliche Abstraktionsschichten einziehen. Aber genau diese können sich im Entwicklungsprozess als Stolperstein entpuppen.

| von JOHN LOUTZENHISER

Dass der Einsatz von Frameworks im Softwareentwicklungsprozess grundsätzlich nützlich und hilfreich ist, haben wir im ersten Teil unserer Artikelserie dargestellt.[1] Dort haben wir beschrieben, wie sie zum einen die Wiederverwendung von Code sowie die flexible Kopplung zwischen Systemen erleichtern können und wie sie zum anderen geeignet sind, die akzidentielle und psychologische Komplexität, die oft mit Softwaresystemen verbunden ist, durch hilfreiche Abstraktionsschichten und die Verkapselung von systemnahen Implementierungsdetails zu minimieren.

Dass der Einsatz von Frameworks durch Abstraktionen auch genau das Gegenteil bewirken und den Softwareentwicklungsprozess immer komplexer werden lassen kann, zeigen wir im zweiten Teil unserer Serie. Dieses Phänomen tritt insbesondere dann ein, wenn ein oder mehrere „Stolpersteine" auf dem Weg liegen.

STOLPERSTEIN ERHÖHTE KOMPLEXITÄT

Die Gefahr einer erhöhten Komplexität durch ein Framework besteht in der Regel immer dann, wenn

- eine Abstraktion, die bestimmte Komplexitäten zu verbergen oder verkapseln versucht, nicht korrekt implementiert wird.
- eine Abstraktion zwar einen wertvollen Werkzeugsatz zur vereinfachten Lösung einer fachlichen oder softwaretechnischen Fragestellung bietet, dazu jedoch Programmierkonzepte (Idiome) und -paradigmen einführt, die hochspezialisiert, kompliziert zu lernen und schwer anzuwenden sind.
- eine Abstraktion zwar für eine losere Kopplung zum Betriebssystem, zur Datenbank oder zum Browser sorgt, aber dafür eine Abhängigkeit zu der spezifischen Technologie oder dem Lieferanten des Frameworks herstellt.

1 .public 01-2015

„Die Softwaretechnik bietet reichlich Gelegenheit für „undichte“ und unbeabsichtigte (akzidentielle) Abstraktionen.“

Lev Gorodinski

STOLPERSTEIN „UNDICHTE“ ABSTRAKTIONEN

Joel Spolsky hat das „Gesetz der undichten Abstraktionen“ formuliert, nach dem alle nicht-trivialen Abstraktionen gewissermaßen „undicht“ sind.

Sogenannte „leaky abstractions“ liegen vor, wenn Frameworks ihre unterliegende Komplexität nur in den meisten Fällen verbergen, die Lösung bestimmter Probleme oder Anwendungsfälle aber erfordert, dass der Programmierer sich mit dieser Komplexität auseinandersetzt – das heißt, die Komplexität auf die Lösungsebene durchschlägt. Diese Undichtigkeit sorgt dafür, dass der Entwicklungsprozess durch Abstraktionen nicht in dem Ausmaß vereinfacht wird wie ursprünglich erhofft oder beabsichtigt.

Moderne Frameworks liefern eine Fülle von Beispielen für undichte Abstraktionen, die die psychologische oder auch die akzidentielle Komplexität erhöhen, besonders wenn sie

- Kenntnisse ihres Abstraktionsmechanismus,
- Kenntnisse der Details der zugrunde liegenden Fachlichkeit oder Systemdetails, die durch die Abstraktion verkapselt werden sollten,
- Erfahrungen mit den Fällen, in denen die Abstraktion beziehungsweise Verkapselung nicht funktioniert,

voraussetzen. Ohne die Abstraktion wäre nur die Kenntnis aus dem zweiten Punkt notwendig gewesen!

Bei der Evaluierung eines Frameworks sind daher folgende Fragen wichtig:

- Sind die Abstraktionen des Frameworks dicht?
- Welche Kenntnisse der zugrunde liegenden (vermeintlich verkapselten) Details sind trotz der Verwendung des Frameworks notwendig?
- Sind Fälle bekannt und dokumentiert, in denen die Abstraktion „undicht“ ist? Muss man damit rechnen, im Projekt mit solchen Fällen konfrontiert zu werden?

STOLPERSTEIN HOHE LERNKURVE

Die Einführung einer Abstraktionsschicht bedeutet in der Regel auch die Einführung neuer Konzepte, Denk- und Arbeitsweisen. Allerdings gibt es keine Garantie, dass diese auch einfacher zu lernen und zu verwenden sind als das, was durch sie abstrahiert werden soll. Die erwarteten Produktivitätsvorteile können durch die Einführung einer Abstraktionsschicht sogar zum Nachteil werden – und zwar dann, wenn ihre Beherrschung mit einer hohen Lernkurve verbunden ist. Und da die durch ein Framework mit hoher Lernkurve erworbenen Kenntnisse und Erfahrungen in der Regel hochspezialisiert und proprietär sind, lassen sie sich auch nur bedingt für andere Projekte nutzen. Für Unternehmen lohnen sich Investitionen in solche speziellen Framework-Kenntnisse daher oft nicht.

Auch und vor allem proprietäre Frameworks können hochspezialisierte Fertigkeiten voraussetzen, wodurch die Wiederverwendbarkeit und Übertragbarkeit der Kenntnisse und Erfahrungen problematisch wird.

Dagegen kann ein Framework, das allgemein bekannt ist und auf üblichen, allgemein bekannten Mustern und Konzepten basiert, sehr gut wiederverwendet und die damit gemachten Erfahrungen auf andere Projekte übertragen werden. So wie das Erlernen einer neuen Programmiersprache ein machbares und realistisches Unterfangen ist, wenn man bereits eine Programmiersprache beherrscht, kann man Design Patterns – das heißt (größtenteils) nichtproprietäre Abstraktionen von Lösungen für gewöhnliche softwaretechnische Probleme – auch leicht in einer anderen Sprache implementieren, wenn man die Konzepte hinter den Entwurfsmustern verstanden und sie bereits in einer bestimmten Programmiersprache implementiert hat.

Bei der Evaluierung eines Frameworks muss daher hinterfragt werden:

„Die Technologie ist ein hilfreicher Diener, aber ein gefährlicher Herr."

Christian Lous Lange

- Wie hoch ist die Lernkurve und wie umfangreich ist der Lernprozess für das Framework im Vergleich zur eingesparten Arbeit?
- Sind die Fertigkeiten, die für dieses Framework benötigt werden, allgemeingültig und wie leicht können Mitarbeiter mit diesen Fähigkeiten gefunden werden?
- Können das Framework und die gelernten Framework-Fähigkeiten in künftigen Software-Projekten wiederverwendet werden und ist der nötige Lernprozess dadurch gerechtfertigt?

STOLPERSTEIN LIEFERANTEN-/TECHNOLOGIE-LOCK-IN

Die Entscheidung für den Einsatz eines Frameworks in einem Prozess ist häufig auch die Entscheidung für einen bestimmten Technologie-Stack. Das heißt, einmal getroffen, lässt sie sich nicht so einfach wieder rückgängig machen. Während die Abstraktionsmechanismen eines Frameworks oft lose Kopplung und Systemflexibilität innerhalb des Frameworks bieten, kann sich der Ersatz eines bereits verwendeten Frameworks selbst als schwierig bis unmöglich erweisen. So bleibt eine Web-Anwendung, die ein ORM[2] -Framework für Persistenz, JSF im Frontend und EJBs in der Anwendungsschicht einsetzt, auf Dauer an diese Technologien gebunden. Einige schwergewichtige Frameworks, wie Oracle ADF, setzen sogar die Verwendung einer ganz bestimmten Version eines speziellen Tools voraus, um das Framework nutzen zu können. Die Aktualisierung der verwendeten Version des Frameworks zieht dann auch eine Aktualisierung dieser Tools nach sich – häufig ein zeitaufwendiges und kostspieliges Unterfangen.

Damit stellen sich bei der Evaluierung eines Frameworks folgende Fragen:

- Macht die Verwendung eines bestimmten Frameworks von einer Technologie oder einem Lieferanten abhängig?
- Machen die Nachteile dieses Technologie-Lock-ins die Vorteile der Flexibilität durch das Framework zunichte?
- Ist ein Lieferanten-Lock-in ein strategischer Vorteil oder eher ein potenzielles Risiko?

2 Object-Relational Mapping = Abstraktionsschicht zwischen objektorientierter Anwendung und relationaler Datenbank

FRAMEWORKS UND ENTWICKLERPRODUKTIVITÄT

Die Einführung von Frameworks ist in der Regel vom Wunsch nach erhöhter Entwicklerproduktivität und leistungsfähigerer Entwicklung getrieben. Es lohnt sich in diesem Zusammenhang also, sich auch mit dem Thema „Entwicklerproduktivität" auseinanderzusetzen. Was genau versteht man unter Entwicklerproduktivität? Kann sie durch ein Framework gesteigert werden? Und wenn ja, wie?

WIE LÄSST SICH ENTWICKLERPRODUKTIVITÄT MESSEN?

„Miss alles, was sich messen lässt, und mach alles messbar, was sich nicht messen lässt." Ob sich dieser Anspruch von Galileo Galilei auch auf die Entwicklerproduktivität übertragen lässt, darf bezweifelt werden. Denn weder die Anzahl der erstellten Code-Zeilen noch die produktiv zum Schreiben von Code benötigten Stunden geben wirklich Aufschluss über die Entwicklerproduktivität.

Ein einfaches, aber komplex und redundant geschriebenes Programm besteht aus vielen Code-Zeilen. Aber weist es auch auf eine höhere Produktivität hin als eine kurze, klare und prägnante Implementierung? Und ist der Entwickler, der diese prägnante Version durch sieben Stunden Nachdenken und eine Stunde Code-Schreiben erstellt hat, nur eine Stunde „produktiv" gewesen oder einen ganzen Arbeitstag? Solche und ähnliche Fragen haben zu der Erkenntnis geführt, dass „Code-Analyse und ähnliche Methoden (...) wenig Aufschluss über das (geben), was ein effektives Software-Team ausmacht".[3]

FOWLER: „PRODUKTIVITÄT LÄSST SICH NICHT MESSEN"

Martin Fowler behauptet, „Entwicklerproduktivität" lasse sich nicht messen[4], und belegt seine These mit der „Function-Point-Analyse". Während ein Entwickler 100 Function Points (FP) in sechs Monaten programmiert, programmiert ein anderer im

3 http://www.infoworld.com/d/application-development/the-futility-developer-productivity-metrics-179244

4 http://martinfowler.com/bliki/CannotMeasureProductivity.html

gleichen Zeitraum nur 30 FP. Doch genau diese 30 FP benötigt der Kunde, um ein weiteres Projekt an den gleichen Auftragnehmer zu vergeben, wodurch weiterer Ertrag und Gewinn generiert werden. Fowler macht darauf aufmerksam, dass die Herausforderung beim Messen der Produktivität darin liegt, den Begriff Output zu definieren und zu quantifizieren. Was gilt bei einem Softwaresystem als angemessener Output? Der Geschäftswert? Der Ertrag? Die eingesparten Kosten? Status und Image in der Branche? Strategische Vorteile? Es gibt keine einheitliche, objektive Berechnungsformel für Produktivität.

ÜBER DEN NUTZEN VON PRODUKTIVITÄTSMASSNAHMEN

Angesichts der oben beschriebenen Schwierigkeiten äußert sich der Softwareberater Steve McConnell skeptisch gegenüber den Möglichkeiten, individuelle Produktivität messen zu können.[5] Seiner Meinung nach gibt es weder einzelne Maßnahmen, die eine verlässliche Bemessung von Produktivität zulassen, noch liefern Maßnahmenbündel Einsichten über die feinen Unterschiede zwischen der Produktivität einzelner Personen. Produktivitätsmaßnahmen sind laut McConnell von geringem Nutzen für die Einschätzung von Projektaufwänden oder -dauer.

SPOLSKY ÜBER SPITZENLEISTUNG

Joel Spolsky führt in die Frage der Messung von Produktivität die Faktoren „Talent“ und „Kreativität“ ein.[6] Echte „Produktivität“, so Spolsky, habe nur wenig mit schnellerem Arbeiten oder vermehrter Erstellung von Code oder FPs innerhalb kürzerer Zeiträume zu tun, sondern vielmehr mit talentierten Spitzenentwicklern, die „die hohen Töne treffen können“ – Entwickler also, die in der Lage sind, „eine Vision von ‚richtig guter Software‘ zu entwickeln und diese dann auch umzusetzen“, die Software entwickeln, die „Stil, Freude und emotionelle Attraktivität“ verkörpert. Frameworks allerdings funktionieren genau gegenteilig. Sie sollen durch Industrialisierung des Entwicklungsprozesses die Notwendigkeit und Bedeutung von Spitzenentwicklern beseitigen helfen.

FAZIT: DAS GESETZ DES ABNEHMENDEN ERTRAGSZUWACHSES

Wenn bei der Suche nach einer Antwort auf die Frage, wie sich Entwicklerproduktivität steigern lässt, der Fokus nur auf Abstraktionen, Frameworks, Technologien gelegt wird, ohne die vielen anderen Faktoren zu berücksichtigten, erinnert das an das „Gesetz des abnehmenden Ertragszuwachses“[7], das besagt: Wenn der Einsatz nur eines Faktors der Produktion erhöht wird, während alle anderen Faktoren unverändert bleiben, steigt der Ertrag zuerst an, dann bleibt er gleich und schließlich sinkt er ab. Möglicherweise hat die Einführung von höheren Programmiersprachen und deren Abstraktionsschichten zunächst zu einem großen Anstieg der Produktivität geführt. Durch das ständige Hinzufügen von immer mehr Abstraktionsschichten ist der Zugewinn allerding kontinuierlich kleiner geworden, bis hin zu einem Absinken der Produktivität. Und die Ursache dafür? Die ist sehr häufig im „Einschleppen“ akzidentieller Komplexität und undichter Abstraktionsschichten zu finden. ●

ANSPRECHPARTNER – JOHN LOUTZENHISER

Senior IT Consultant

Public Sector

- +49 173 859 4235
- john.loutzenhiser@msg-systems.com

5 http://www.joelonsoftware.com/articles/HighNotes.html
6 http://www.construx.com/10x_Software_Development/Measuring_Productivity_of_Individual_Programmers
7 https://de.wikipedia.org/wiki/Ertragsgesetz