

# Modularisierung komplexer Softwaresysteme

Claudia Pohle und Dr. Christian Hock

Besonders erfolgreiche Unternehmen kennen das Szenario: Eine innovative Software, in den 1990er Jahren von einigen genialen Köpfen ins Leben gerufen, anschließend über 10 Jahre engagiert weiterentwickelt und erfolgreich verkauft, stößt an die Grenzen der Wartbarkeit und Erweiterbarkeit. Die nüchternen „Eckdaten“ lauten nicht selten: 3 Mio. Lines of Code, mehrere eingesetzte Programmiersprachen, einige 100 DLLs, diverse Komponententechnologien, und zu allem Übel passt auch die Dokumentation nicht zur Implementierung. Verschärfend wirkt, dass viele Know-how-Träger von „damals“ nicht mehr verfügbar sind und wichtige Kunden zeitnah neue Funktionen benötigen – um damit in den Einsatz zu gehen. Auch wenn es DIE EINE LÖSUNG nicht gibt – die SYSTEMATISCHE MODULARISIERUNG stellt ein besonders wirksames und weitreichend nutzbringendes Rezept dar.

(Abbildung: istockfoto.com/Olga Popova)

Wenn bei Abschätzungen für das Hinzufügen einer eigentlich kleineren, scheinbar abgeschlossenen Funktionalität unglaublich hohe Aufwände genannt werden, kommt auf Nachfragen oft die Antwort: „Das hat Auswirkungen auf mehrere andere Komponenten“, obwohl diese nicht mit der neuen Funktion zusammenhängen. Oder die Behebung eines Fehlers verursacht plötzlich eine Verhaltensänderung an einer ganz anderen, nicht damit im Zusammenhang stehenden Stelle. Oder der Systemtest lässt sich nicht starten, weil einzelne Anteile der Software noch nicht fertig gestellt sind, zum Start aber alles benötigt wird. Oder der Austausch von Hardware wird um Größenordnungen teurer als geplant, weil Veränderungen in Hardware-fernen Bestandteilen vorgenommen werden müssen.

Dies alles sind Anzeichen, dass die Software nicht modular genug aufgebaut ist. Was aber bedeutet Modularität in der Software? Warum ist modular aufgebaute Software besser als nichtmodular aufgebaute? Worin ist sie besser?

## Autoren:

Claudia Pohle ist Senior Software-Ingenieurin, Dr. Christian Hock ist Bereichsleiter Industry bei Berner & Mattner Systemtechnik GmbH.

## Was bedeutet Modularität in der Software?

Modular aufgebaute Software bedeutet, dass die gesamte Software aus verschiedenen Bausteinen aufgebaut ist, wobei die Bausteine besondere Eigenschaften besitzen:

- Jeder Baustein fasst Daten und Funktionen zusammen, die eine abgeschlossene Aufgabe erfüllen.
- Jeder Baustein kommuniziert mit den Nachbarbausteinen über genau definierte Schnittstellen.
- Deren Benutzung setzt keinerlei Kenntnis über die Interna voraus. Bausteine mit gleichen Schnittstellen können gegeneinander ausgetauscht werden, ohne dass Änderungen an anderen Bausteinen vorgenommen werden müssen.
- Bausteine können auch ohne Kenntnis der inneren Abläufe in Systeme integriert werden. An ihren Schnittstellen sind die Bausteine testbar, ohne dass andere Bausteine zu diesem Zeitpunkt integriert sein müssen.

## Vorteile

Einige Vorteile einer auf diese Art aufgebauten Software sind schnell ersichtlich: einzelne Bausteine sind gegen andere austauschbar, sie sind einzeln in ihrer

Funktionalität gut testbar. Bei der Integration sind „verspätete“ Bausteine durch Dummies ersetzbar, die einfach nur die Schnittstellen bedienen und so die Integration anderer Teile nicht verzögern.

Wenn nur einzelne Bausteine innerhalb eines Systems geändert wurden (aber nicht deren Schnittstellen!), reichen Tests des Bausteines aus, um sicherzustellen, dass sich die Gesamtfunktionalität nicht verändert hat. Da der Test einen erheblichen Anteil an den Kosten eines Software-Produktes ausmacht, ist dies ein wichtiger Vorteil.

Aufgrund der Austauschbarkeit der Bausteine ist auch die Wartung eines Bausteins mit klar definierten Aufgaben und Schnittstellen einfacher und kostengünstiger. Nicht zuletzt lassen sich Gesamtprodukte und Produktvarianten auf Baustein-Basis leichter zusammenstellen, testen und beherrschen.

Wenn einige der zu lösenden Aufgaben des Software-Systems sicherheitsrelevant sind und andere nicht, ist es durch einen streng modularen Aufbau der Software möglich, den durch die Sicherheitsrelevanz erforderlichen Mehraufwand auf die sicherheitsrelevanten Bausteine zu beschränken. Bei einer verteilten Entwicklung, sei es mit unterschiedlichen Partnern oder an

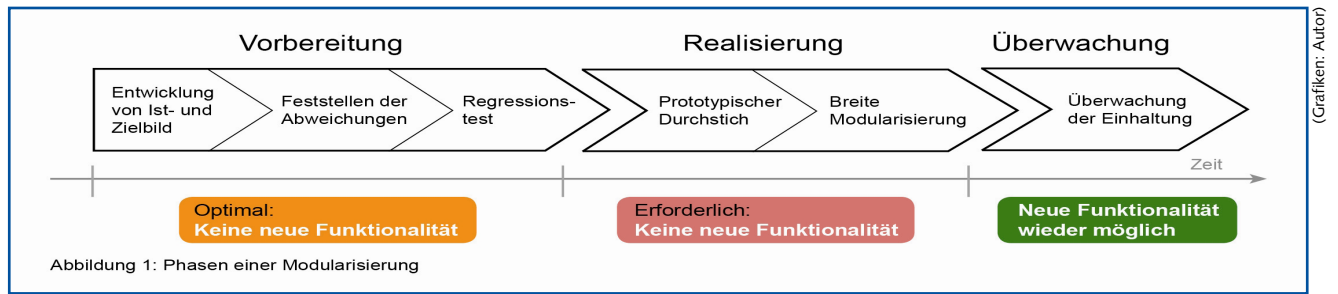


Abb. 1: Phasen einer Modularisierung

verschiedenen Standorten, sind die Steuerung der Entwicklung, die Integration und der Test wesentlich einfacher, wenn die Verteilungsgrenzen mit den Bausteingrenzen übereinstimmen.

### Vorgehensweise bei der Modularisierung

Aber wie ist ein solcher Zustand der Software zu erreichen, wenn nicht neu angefangen werden kann, sondern auf einer vorhandenen komplexen und verwobenen Software aufgebaut werden soll? Klar ist: eine Modularisierung kostet Geld und Zeit. Geld, weil Arbeiten durchgeführt werden, die keinem Kundenfeature zugeordnet werden können, und Zeit, weil diese Arbeiten geplante Kundenfeatures verschieben. Deshalb ist es zunächst einmal wichtig, einen günstigen Zeitpunkt für die Modularisierung zu finden, also wenn sowieso keine neuen Features geplant sind oder eine zeitliche Verschiebung der Features keine gravierenden Auswirkungen hat. Abbildung 1 zeigt die Phasen eines Modularisierungsprojektes. Zumindest in der Realisierungsphase der Modularisierung sollte kein Funktionshub geplant werden. In der Vorbereitungsphase ist dies eingeschränkt noch möglich.

### Vorbereitung

Eine Modularisierung startet mit der Vorbereitungsphase. Sie beinhaltet die Entwicklung eines Zielbildes (aus welchen Bausteinen soll das System bestehen?), die Dokumentation des Istbildes (Abbildung 2) und die Identifikation der Abweichungen (an welchen Stellen muss etwas getan werden?). An diesem Prozess müssen Experten beteiligt sein, die das vorhandene System mit seinen Stärken und Schwächen sehr gut kennen – erfahrene Software-Architekten

und auch Systemingenieure – damit keine Funktionalität verloren geht.

**Zielbild – Zielarchitektur:** Meist gibt es bereits ein Bild davon, aus welchen Bausteinen die Software bestehen soll; es wurde am Projektanfang erstellt und (mehr oder weniger) bei Veränderungen aktualisiert. Nun muss geprüft werden, inwieweit dieses Bild mit dem derzeitigen Funktionsumfang übereinstimmt und ob die damals erdachte Aufteilung immer noch sinnvoll ist. Dies liegt vor allem in der Verantwortung der Software-Architekten und Systemingenieure. Durch eine funktionale Analyse und eine Schnittstellenanalyse werden abgeschlossene Aufgabenblöcke gefunden, die zu einem Zielbild zusammengesetzt werden. Bei Systemen, die in mehreren Varianten existieren, muss

das Variantenmanagement bereits in den Anforderungen berücksichtigt sein. Ist dies noch nicht der Fall, muss es vor der Zuordnung zu Aufgabenblöcken erfolgen. Bei der Festlegung der Aufgabenblöcke ist auch die Testbarkeit zu berücksichtigen. Es muss sichergestellt werden, dass die einzelnen Bausteine autonom testbar sind, idealerweise automatisiert.

**Istbild:** Wenn eine aktuelle und vollständige Dokumentation des bestehenden Systems vorhanden ist, wird das Istbild daraus erstellt. Häufig stimmen jedoch Dokumentation und Software nicht überein. Dann kommen die Kenner der alten Software ins Spiel, die – durch statische Code-Analysen unterstützt – das System betrachten und das Istbild erstellen.

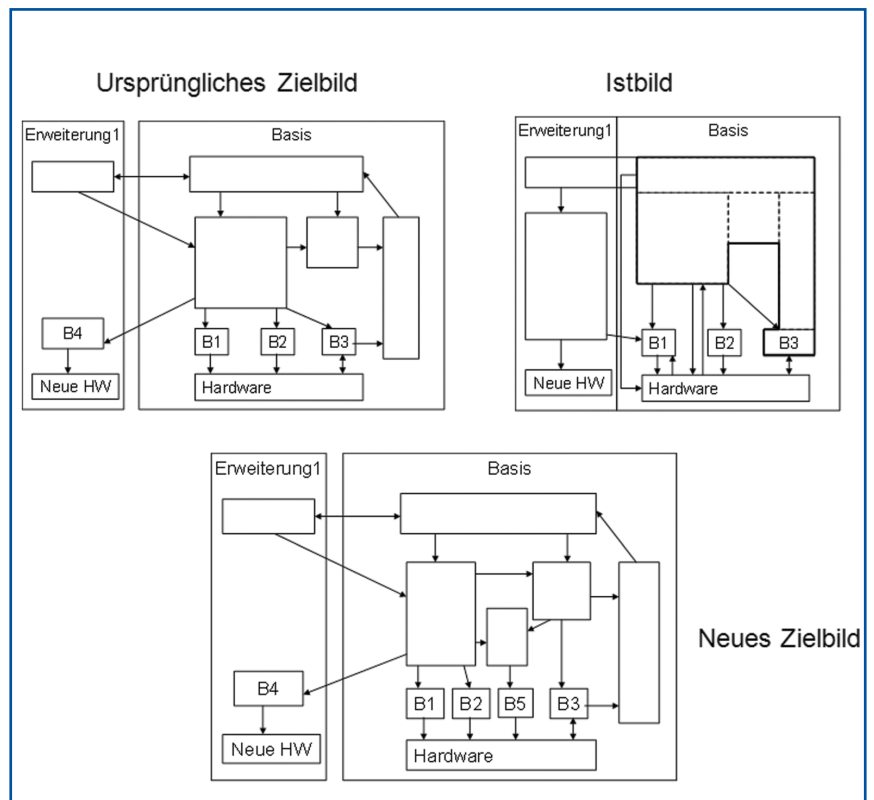


Abb. 2: Ergebnisse der Vorbereitungsphase

**Abweichungen:** Um die notwendigen Veränderungen festlegen zu können, ist festzustellen, an welchen Stellen die derzeitige Software nicht mit dem Zielbild übereinstimmt.

Interessant ist auch herauszufinden, warum davon abgewichen wurde. Denn oft hat die Abweichung einen triftigen Grund. So kann sich herausstellen, dass z. B. ein Baustein gefehlt hat, dessen Aufgaben nun verschiedene andere mit übernehmen müssen.

Wenn geklärt ist, was zu tun ist, muss der Aufwand für die Realisierung und den Test der Modularisierung abgeschätzt werden. Dabei ist eine Etablierung systematischer Tests der Bausteine an ihren Schnittstellen mit einzurechnen.

Um sicherzustellen, dass das System nach der Modularisierung das exakt gleiche Außenverhalten wie vorher hat, sind Regressionstests auf Systemebene enorm wichtig. In diesen Tests wird das Verhalten der alten Software dokumentiert und kann somit als Referenz für die modularisierte Version herangezogen werden. Wenn solche Tests nicht oder nicht ausreichend vorhanden sind, sind sie in dieser Phase zu erstellen. So lassen sich Verhaltensabweichungen vermeiden, die Kunden irritieren.

## Realisierung

Für die Realisierung selbst ist – bevor sie in die Breite geht – ein prototypischer Durchstich, also die Modularisierung z. B. einer kleineren Funktionalität, sinnvoll. Hieraus lassen sich Erkenntnisse über den Aufwand und die Schwierigkeiten oder Risiken gewinnen, die in die Aufwandsschätzung einfließen.

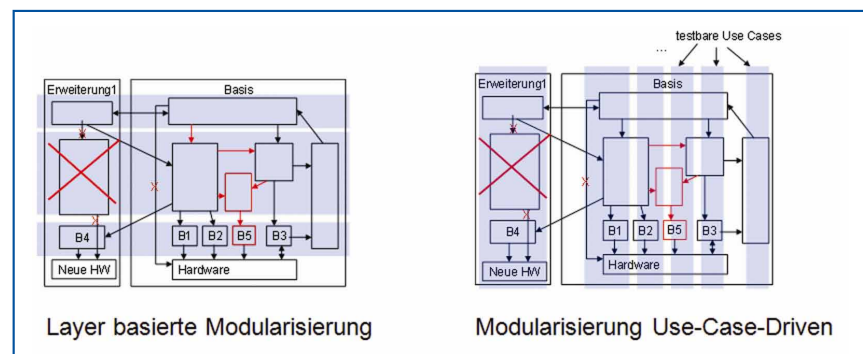
Nun kann eine breite Modularisierung beginnen. Dabei müssen die Schnittstellenklassen zur Verfügung gestellt und von den anderen Bausteinen genutzt werden; die Interna der Bausteine auf die neuen Schnittstellenklassen müssen angepasst und automatisierte Tests der Baustein-Schnittstellen implementiert werden. In welcher Reihenfolge modularisiert wird, ist projektabhängig. Es kann sinnvoll sein, Use-Case für Use-Case vorzugehen. Dies hat den Vorteil, dass nach jedem Use Case eine test-

bare Gesamtfunktionalität vorhanden ist (Abbildung 3). In einem stark von Layern (Schichten) bestimmten Projekt ist möglicherweise ein layerweises (horizontales) Vorgehen sinnvoll. Die Realisierung ist in jedem Fall durch Software-Analyse-Tools zu überwachen, um ein Erreichen der gewünschten Ziele sicherzustellen.

Das durchführende Team muss den gesamten Modularisierungsprozess in seiner Wichtigkeit erkennen und akzeptieren. Ist das erforderliche Know-

- sich Zeiten für Fehlersuche und -behebung durch klar definierte Schnittstellen und einzeln testbare Bausteine verkürzen,
- das Variantenmanagement erleichtert wird und
- der Austausch einzelner Bausteine (z. B. wegen Hardwarewechsel) ermöglicht wird, ohne das Gesamtsystem zu beeinflussen.

Eine Modularisierung durchzuführen ist eine anspruchsvolle Aufgabe, die Geld kostet. Das Initiieren, Durchführen und



**Abb. 3: Unterschiedliche Vorgehensweisen bei der breiten Realisierung**

how nicht vorhanden, sind die Teammitglieder ggf. zu schulen. Außerhalb des Teams muss diese Arbeit hohe Anerkennung finden, auch wenn kein direkter Projektfortschritt im Sinne von Kundenfeatures erkennbar ist.

## Überwachung

Die Überwachung der Einhaltung der Modularität endet nicht am Ende des Modularisierungsprozesses, sondern geht im Gegenteil verstärkt auch in den nachfolgenden Projektphasen weiter, um nicht bereits erzielte Fortschritte wieder zu verlieren. In diesem Sinne ist es auch wichtig, dass alle Mitarbeiter – nicht nur die, die die Modularisierung durchgeführt haben – das Ziel einer modularen Software und deren Vorteile kennen und sich damit identifizieren.

## Fazit

Eine Modularisierung von komplexen, über Jahre gewachsenen Systemen ist aus finanziellen Gründen sinnvoll, da

- durch verbesserte Wartbarkeit Funktionalitätsänderungen kostengünstiger werden,

Begleiten eines Modularisierungsprozesses erfordert die aktive Unterstützung des Managements. Allein einen Zeitpunkt zu finden, bei dem eine Unterbrechung der Kundenfeature-Entwicklung akzeptabel ist, kann problematisch sein. Ein Modularisierungsprojekt ist sorgfältig zu planen und zu realisieren. Alle Beteiligten (auch die Kunden) sollten dies als wichtiges und lohnendes Projekt betrachten. Die daran beteiligten Mitarbeiter und Mitarbeiterinnen müssen sich mit dem Ziel identifizieren und das nötige Know-how besitzen. Ggf. sind Schulungs- und Coaching-Maßnahmen durchzuführen. Risiken lassen sich durch einen prototypischen Durchstich (z. B. anhand eines Use Cases) frühzeitig erkennen und durch Maßnahmen abschwächen. Selbst wenn die eigentliche Realisierungsphase abgeschlossen ist – d.h. das komplexe System in klar definierte Bausteine zerlegt und zusammengebaut wurde – ist die Einhaltung der hart erarbeiteten Bausteingrenzen in der Weiterentwicklung des Systems zu überwachen.

Bei langlebigen, komplexen Systemen eine Modularisierung durchzuführen, ist teuer. Sie nicht durchzuführen, ist teurer. Es lohnt sich. ■