Alptraum Legacy Code – Wie gehen Profis damit um?

Legacy Code bringt oft gewaltige Probleme bei der Entwicklung neuer Features mit: Redundanzen, nichtssagende Variablen oder ausufernde Methoden sind nur einige Hürden. Wie geht man damit um?

LARS JACOBI *

Jeder, der im Umfeld der Software-Entwicklung agiert, hat diesen Begriff schon gehört und musste sich vermutlich auch schon mit den Problemen beschäftigen, die Legacy Code mit sich bringt. Doch wie definiert sich Legacy Code genau? Verschiedene Experten und Plattformen sind da oft unterschiedlicher Auffassung. Für diesen Artikel wird der Einfachheit halber jeder Code als Legacy Code bezeichnet, der eines oder mehrere der folgenden Kriterien erfüllt:

- Der Code ist nicht durch automatisierte Tests abgesichert.
- Der Code entspricht nicht den aktuellen „Regeln der Kunst“ (Clean Code).
- Die Entwickler scheuen sich, den Code zu verändern oder zu erweitern („Never touch a running system”).
- Keiner versteht den Code mehr wirklich.

*** Lars Jacobi**
... ist Softwareentwickler bei Method Park in Erlangen.

Entstehung und Konsequenzen von Legacy Code

Legacy Code entsteht meist schleichend, über einen langen Zeitraum hinweg. Der Schlüsselaspekt hierbei ist die Häufigkeit der gewissenhaften Wartung: Eine Maschine muss gewartet werden – Programmcode bildet da keine Ausnahme. Leider wird dies für kurzfristige Gewinne und neue Projektziele immer wieder übergangen, in Führungsetagen oft sogar komplett ausgeblendet. Legacy Code wird auch durch Entwickler begünstigt, die sich nicht weiterbilden, zu wenig Aufwand in das Testen ihres Codes stecken und an alten Standards festhalten. Selbst disziplinierte Entwickler schreiben leicht mal schlechten Code, wenn bereits schlechter Code in ihrem direkten Umfeld existiert.

Schwerwiegende technische Folgen von Legacy Code verhärten sich zu unwartbarer, unerweiterbarer und untestbarer Software. Dabei richtet Legacy Code nicht nur auf technischer Ebene, sondern auch im Team erhebliche Schäden an: Der Lieferdruck steigt, genauso wie der Frust über die Handlungsunfähigkeit. Die Projektkosten explodieren, die Aufwände sind nicht mehr akkurat schätzbar, die Probleme werden in sämtlichen Bereichen des Unternehmens spürbar. In manchen Fällen gelten 30% Mehrkosten als On-Top-Budget als eine feste kalkulatorische Größe für Software-Qualität.

Irrtümer zum richtigen Umgang mit Legacy Code

Wie aber geht man mit Legacy Code um? Soll man das existierende Projekt aufgeben, den Code verwerfen und neu schreiben? In den meisten Fällen lautet die Antwort: Nein! Das Problem lässt sich auch langfristig nicht effektiv lösen, indem man das Projekt neu aufsetzt. Man kann kein Produkt mit den aktuell vorhandenen personellen Ressourcen warten und parallel dazu ein Neues entwickeln. Selbst wenn das Produkt noch nicht auf dem Markt ist und die Führungsriege die erforderlichen Mittel zum Reset freigibt, führt ein Neu-Aufsetzen des Projektes in der Regel unwiderruflich zu neuem Legacy Code.

Der Grund: Viele Software-Entwickler halten an alten Vorgehensweisen und Techniken fest und sind gegenüber Veränderungen am Alt-Bewährten oft misstrauisch. Das führt dazu, dass alte Probleme am Ende oft nur neu verpackt werden. Diese Betriebsblindheit bildet in einem neu aufgesetzten Projekt den Nährboden für neuen Legacy Code.

Refactoring: der Anfang vom Ende des Legacy Codes

Will man Legacy Code Herr werden, muss man investieren. Zunächst muss das Personal geschult werden: Wie soll Code nach den aktuellen Standards geschrieben und getestet werden? Ab wann spricht man von Clean Code? Nach Robert C. Martin, Autor des

Buchs „Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code“, gelten hier die fünf sog. SOLID-Prinzipien:

- Single responsibility principle (SRP)
- Open/closed principle (OCP)
- Liskov substitution principle (LSP)
- Interface segregation principle (ISP)
- Dependency inversion principle (DIP)

Der Clean-Code-Ansatz ist hierbei keine einmalige Lektüre, sondern eine Mentalität, die gelebt und stets erneuert werden muss. Es wird ein Gefühl für Ästhetik entwickelt, das dazu motiviert, schönen Code zu schreiben, der lesbar und wartbar ist. Software-Entwickler, die diesen Ansatz verinnerlicht haben, sehen in ihrem Code nicht nur Funktionalität, sondern auch Kunst. Das Ergebnis ist dabei nicht nur Code, sondern auch ein Erfolgsgefühl und eine Art von Stolz, die den Blick für „unsauberen“ Code schärft.

Ist das erst in den Köpfen und Herzen der Entwickler angekommen, muss Zeit für die Qualitätssicherung der Anwendung eingeplant und freigegeben werden. Software-Qualität entsteht durch Refactoring von Quellcode. Refactoring ist eine Technik zum Restrukturieren von Quellcode, ohne das Verhalten des Codes zu verändern. Vor allem Refactoring im Team bietet hervorragende Möglichkeiten mit Methodiken, wie etwa Pair Programming, die Qualität der Software zu verbessern. Zugleich tauschen die Teammitglieder so auch Wissen und Know-how aus.

Professionelles Refactoring verfolgt einen testgetriebenen Entwicklungsansatz (Test Driven Development, TDD). Diese durch einen Test-First-Ansatz charakterisierte Methodik sieht vor, dass Code, der geändert oder hinzugefügt werden soll, zuvor durch automatisierte Tests abgedeckt ist. Diese Tests sollten in Form von Unit Tests oder Integration Tests vorliegen und müssen – bezogen auf Legacy Code – keinen fachlichen Anforderungen entsprechen, sondern das tatsächliche Verhalten der Anwendung sichern! In fast jedem Legacy-System ist wichtiger, was das System tut, als das, was es tun soll.

Ist einmal ein solides Sicherheitsnetz gespannt, kann man sich mit den eigentlichen Änderungen im Code befassen. Hier gilt die Devise zunächst kleine Änderungen vorzunehmen und dann über die Tests die bisherige Funktionalität des Codes sicherzustellen: Ändern -> Testen -> Ändern -> Testen

Diese Methode wird auch „Cover and Modify“ genannt. Sie stellt eine solide Vorgehensweise dar, um Legacy Code Stück für Stück verständlicher und wartbarer zu machen. Dafür müssen aber unbedingt sogenannte charakteristische Tests vorhanden sein. Schreiben Sie keine, verfolgen Sie einen Ansatz, den Michael C. Feathers in „Working Effectively with Legacy Code“ als „Edit and Pray“ bezeichnet: Sie planen Ihre Änderungen und hoffen nach dem Check-in, dass Sie nichts kaputt gemacht haben. Dieses Vorgehen sollten Sie vermeiden.

Effektiver Schutz vor Legacy Code

Hier stellt sich die Frage: Wie schützt man das Produkt vor Legacy Code? Martin hat dazu die sogenannte „Boy Scout Rule“ aufgestellt. Diese wirbt im Kern um ein simples Verhalten: Lassen Sie den Code sauberer zurück, als Sie ihn vorgefunden haben. Es ist eben nicht genug, sauberen Code zu schreiben, man muss ihn auch über einen längeren Zeitraum hinweg pflegen. In Zeiten, in denen Software durch neue Anforderungen stetig wächst, hat die Wartung besonderes Gewicht. Wenn sich jeder Entwickler an diese simple Regel hält, wird Legacy Code nicht nur aktiv vermieden, auch die Code-Qualität wird langsam aber stetig verbessert.

Eine solide Testabdeckung ist für Software-Projekte essentiell. Ein Projektteam kann und muss sich effektiv gegen Legacy Code schützen, indem es jedes neu implementierte Feature durch automatisierte Tests absichert. Auch Code Reviews tragen durch ihr Vier-Augen-Prinzip erheblich zur Prävention bei. Diese Methodik gilt in vielen Software-Projekten sogar als das Quality Gate schlechthin.

Legacy Code ist der flüssige Teer in der Natur eines jeden Software-Projektes. Er führt dazu, dass die Entwicklung immer langsamer, Änderungen immer riskanter und die Teammitglieder immer frustrierter werden. Im schlimmsten Fall führt Legacy Code zum Ruin eines Unternehmens. Das Säubern von Legacy Code zu Clean Code ist mit viel Arbeit verbunden. Aber eines ist klar: Es ist ein notwendiges Übel. Hat man diesen Weg einmal eingeschlagen, dann lohnt er sich. Die Software-Qualität steigt, die Kunden sind zufriedener und die Entwicklungsteams können effektiv arbeiten. // **SG**

Method Park