Kapitel 1

Codequalität in einer agilen Welt

In diesem Kapitel:

»Wir sind, was wir jeden Tag tun. Perfektion ist kein einmaliger Akt, sondern eine Gewohnheit.«

Aristoteles

Als Softwareentwickler möchten wir möglichst hervorragende Produkte schaffen. Die Benutzer sollen Freude an unseren fehlerfreien Programmen haben, und es sollte gewährleistet werden, dass die Programme von höchster Qualität sind. Unter unseren Kollegen bemühen wir uns darum, unsere technischen Fähigkeiten mit elegantem und stabilem Code zu demonstrieren. Jeder von uns arbeitet hart, um diese Ziele jeden Tag zu erreichen. Unsere stetigen Anstrengungen bei der Entwicklung hochqualitativer Software in Verbindung mit unserem immer währenden Streben nach der Verbesserung unserer Methoden und Verfahren festigen unsere Gewohnheiten als Entwickler. Wie Aristoteles andeutet, werden Spitzenleistungen durch eine konstant gute Ausführung erzielt.

Die Herstellung hochqualitativer Software ist allerdings eine schwierige Aufgabe. Sogar die rudimentärsten Programme weisen Fehler auf. Auch die besten Entwickler hinterlassen Bugs in ihren Programmen. Menschen sind von Natur aus unvollkommen und machen mitunter Fehler. Wenn Entwickler von Menschen gegebene Anweisungen in die beauftragte Softwareanwendung übersetzen, unterlaufen ihnen Fehler, es kommt zur Auslösung von Ausnahmen und es werden auch qualitative Probleme auftreten.

Aber es wäre unfair, die Verantwortung für die Softwarequalität allein den Entwicklern aufzuerlegen. Softwareentwicklung ist ein Prozess, bei dem viele Beteiligte aus unterschiedlichen Disziplinen einbezogen sind. Bei Microsoft werden beispielsweise Entwicklerteams typischerweise in drei Disziplinen aufgeteilt: Programmmanagement, Entwicklung und Test. Programmmanager stellen sicher, dass die Produktspezifikationen präzise und gut durchdacht sind, und setzen klar umrissene Ziele für die Qualität des Endprodukts. Entwickler erstellen Designs mit maximaler Effizienz und Flexibilität, gewährleisten die Genauigkeit der Algorithmen und setzen die besten wohlbekannten Codeimplementierungsverfahren wirksam ein. Tester berücksichtigen jede mögliche Kombination, sowohl was das Verhalten des Codes als auch die Anwendung betrifft, und überprüfen und durchlaufen jeden Codepfad der Software. Der Qualitätsaspekt ist ein durchgehender Bestandteil des gesamten Entwicklungszyklus und liegt daher in der Verantwortung jedes Beteiligten. Softwareunternehmen verstehen dieses Prinzip und bemühen sich sehr darum, Prozesse und Programme so zu implementieren, dass sichergestellt ist, dass jedes Teammitglied das Hauptaugenmerk auf Qualität setzt.

Softwareentwicklungsprozesse und -methoden existieren in der Industrie seit den späten 1960er Jahren. Als die Rechenleistung anwuchs, wurde die Software im zunehmenden Maße immer komplexer. Diese gesteigerte Komplexität bei der Softwareentwicklung und die Tatsache, dass die Industrie in dieser Hinsicht noch unreif war, führten bei Softwareprojekten zu verschiedenen Problemen, einschließlich Kostenüberschreitungen, Software von geringer Qualität infolge fehlender qualitativer Prozesse sowie Code mit geringer Wartungsfreundlichkeit. Die Folge war, dass formale Softwareentwicklungsprozesse ins Leben gerufen wurden. Deren Hauptziel lag darin, eine Reihe von Aufgaben mit einer formalen Struktur zu versehen, die letztendlich zu einem verbesserten Ergebnis der Entwicklungsbemühungen führten. Einfach ausgedrückt, bestand das Ziel der formalen Entwicklungsprozesse darin, ein Produkt zu erzielen, das von höherer Qualität und in der tatsächlichen Markteinführung bzw. Verteilung besser planbar ist. Die Verwendung eines Prozesses zur Sicherstellung der Qualität eines Produkts ist bei der Herstellung nahezu jedes Produkts und Dienstes von entscheidender Bedeutung, unabhängig vom jeweiligen Industriezweig. Gelegenheiten dafür ergeben sich für Sie als Softwareentwickler, wenn Sie Erfahrung mit einer oder mehreren verschiedenen Projektmanagementmethoden in Ihrer beruflichen Laufbahn gesammelt haben, etwa mit der Wasserfallmethode, agilen Methoden und auch mit dem Microsoft Solutions Framework (MSF).

Herkömmliche Methoden in der Softwareentwicklung

Sie haben vermutlich schon den Ausdruck *Wasserfall* gehört, der auf den herkömmlicheren Softwareentwicklungszyklus (*software development life cycle – SDLC*) angewendet wird. Der Ausdruck *Wasserfall* bezieht sich auf die Weise, mit der jede Phase des SDLC stetig und sequenziell nach unten läuft (Abbildung 1.1). Bei den einzelnen Phasen handelt es sich um folgende:

- **Anforderungen** – eine detaillierte Beschreibung der herzustellenden Software
- **Design** – der Prozess, bei dem die Planung der Implementierungsdetails eines Softwareprojekts erfolgt
- **Implementierung** – der Prozess von Programmierung, Test und Debugging der einzelnen Bestandteile der zu entwickelnden Software
- **Integration und Prüfung** – die Phase, in der die unterschiedlichen Bestandteile der Software zusammengebracht werden, um sie zu integrieren und einem breiteren Test zu unterziehen
- **Installation und Wartung** – der Abschluss des SDLC, bei dem die Software verteilt wird und in einen Zyklus von nachträglichen Korrekturen und Anpassungen übergeht

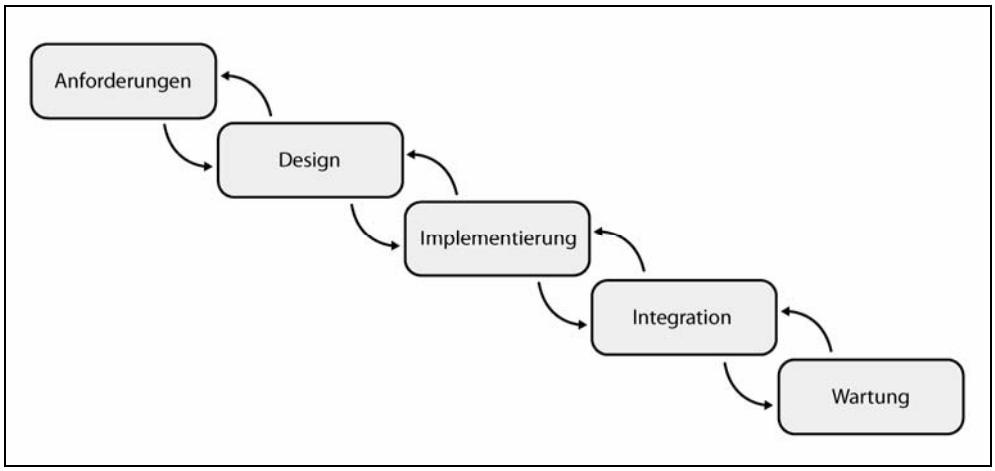

Abbildung 1.1 Herkömmlicher Softwareentwicklungszyklus

Die Starrheit des Modells erfordert bis zur Fertigstellung der Software einen vorwärts gerichteten, sequenziellen Ablauf durch diese Phasen und setzt voraus, dass zunächst jede Phase abgeschlossen werden muss, bevor die nächste beginnen kann. Die Anforderungsspezifikationen müssen vollständig fertiggestellt werden, bevor mit dem Design fortgefahren werden kann. Sobald das Design abgeschlossen und ein Plan vorhanden ist, der den Entwicklern zur Implementierung der Anforderungen dient, beginnt die Programmierung. Die Integration folgt der Programmierung, und die verschiedenen Teile des Programms werden zusammengebracht und als Gesamtsystem getestet. Wenn die Integration abgeschlossen ist und die Software verteilt wird, startet die Wartungsphase. In jeder dieser Phasen wird außerdem eine große Betonung auf schriftliche Dokumentation gelegt, wodurch Wasserfallprojekte zu schwergewichtigen Projekten neigen.

Das Wasserfallmodell geht den Qualitätsanspruch mit der Empfehlung an, dass, je mehr Zeit bei Analyse und Design der Software verbracht wird, desto höher der Anteil der Fehler ausfällt, die bereits in einem frühen Prozess aufgedeckt und in der anschließenden sich nach unten fortsetzenden Entwicklung des

Codes vermieden werden. Kritiker wenden ein, dass diese Methode nicht gut in der Praxis funktioniert, da kritische Details des Programms oder der Anwendung nicht wirklich bekannt sein können, bevor das Entwicklungsteam mit der Implementierung Fortschritte macht. Da das Modell außerdem zwingend die Fertigstellung der Spezifikationen vor dem Beginn der Programmierung erfordert, gibt es nur sehr begrenzte Möglichkeiten, ein frühes Feedback von Kunden oder Betatestern in den Releasezyklus zu integrieren, wodurch das Team bei der zeitnahen Berücksichtigung von nachträglichen Kundenanforderungen stark eingeengt wird. Schlussendlich tritt ein durchgehender Test erst zu in einem späten Zeitpunkt des Zyklus in Erscheinung. Dadurch wird die Qualität der Software gefährdet, sobald mit der Integration von kritischen Elementen begonnen wird.

HINWEIS Die Wasserfallmethode wird häufig Dr. Winston W. Royce zugeschrieben, der 1970 ein Papier mit dem Titel *Managing the Development of Large Software Systems* (wörtlich: *Bewerkstelligung der Entwicklung von großen Softwaresystemen*) verfasste. Die Ironie dieser zweifelhaften Ehre ist, dass es Dr. Royce selbst war, der darauf hinwies, dass diese Methode gefährlich sei und zu Fehlern einlade. Er führte aus, warum das Testen am Ende des Entwicklungszyklus, bei dem alle kritischen Komponenten des Systems erstmals zusammengeführt werden, nicht realistisch sei. Er setzte seine Ausführungen mit dem Hinweis fort, dass in einem späten Zyklus aufgedeckte Fehler zu unausweichlichen Kostenüberschreitungen infolge der hohen Wahrscheinlichkeit nachträglicher Designänderungen führen. Obwohl er an die konzeptuelle Idee des Mehrphasenzyklus glaubte, wünschte er sich eine Entwicklung hin zu einem stärker iterativen Ansatz, der wahrscheinlich ein besseres Ergebnis als das sequenzielle Modell erzielen würde.

Befürworter und Gegner einmal beiseite genommen, hat die Wasserfallmethode des Softwareprojektmanagements ein Gerüst für viele davon abgeleitete und heute existierende Modelle bereitgestellt. Ebenso wie die späten 1980er und frühen 1990er Jahre die Entwicklung von Rapid Application Development-Methoden (RAD) hervorbrachten, haben sich viele Unternehmen in den letzten Jahren in Richtung agiler Entwicklungsmethoden bewegt. Beide sind Derivate der Wasserfallmethode. Diese neueren Modelle sind in ihrem Ansatz in Hinblick auf den Softwareentwicklungsprozess erheblich iterativer und sie treiben kürzere, schnellere und zielstrebigere Releasezyklen voran.

Agile Methoden bei der Softwareentwicklung

Die Bewegung hin zu schlanken Softwareentwicklungsmethoden ist auf die allgemeine Unzufriedenheit mit schwergewichtigen, restriktiven Methoden wie der Wasserfallmethode zurückzuführen. Einige dieser schlanken Methoden kamen ab Mitte der 1990er Jahre auf und betonten Konzepte wie selbstorganisierende Teams, Kommunikation von Angesicht zu Angesicht, schlanke Dokumentation und häufige Releases. Schließlich stellten einige Erschaffer dieser Methoden, darunter Kent Beck, Martin Fowler und Ward Cunningham, die Agile Alliance[1] auf und vereinten die Prinzipien der agilen Softwareentwicklung mit dem Agile Manifesto.[2]

[1] *http://www.agilealliance.org* (engl.)

[2] *http://www.agilemanifesto.org/principles.html* (engl.)

Agile Methoden neigen zur Förderung iterativer Ansätze bei der Softwareentwicklung unter Einsatz kleiner, selbstorganisierender Teams, die in kurzen, stark von Kooperation geprägten Releasezyklen arbeiten. Die Qualität steht in jeder Phase der agilen Softwareentwicklung an erster Stelle, und viele der zentralen Prinzipien – wie Paarprogrammierung (*pair programming*), testgetriebene Entwicklung, Refactoring und kontinuierliche Integration – stellen sicher, dass Fehler schnell und oft noch innerhalb des Releasezyklus gefunden und beseitigt werden, im Unterschied zu den herkömmlicheren Methoden.

Es sind heutzutage viele verschiedene agile Methoden im Einsatz – etwa Scrum, extreme Programmierung (als XP abgekürzt), testgetriebene Entwicklung (*test-driven development, TDD*), Refactoring, kontinuierliche Integration und weitere. Des Weiteren gibt es eine ganze Menge hervorragender Quellen, die sehr hilfreich sind, um die Prinzipien und Verfahren der einzelnen Methoden zu verstehen (vergleiche die Literaturübersicht in Anhang A). Es lohnt sich, einige der bekannteren agilen Methoden in diesem Kapitel zu untersuchen und zu verdeutlichen, welche der vorgestellten Prinzipien und Verfahren zur qualitativen Steigerung auf einen längerfristigen und erfolgreichen Einsatz in den Entwicklungsabteilungen von Microsoft zurückblicken können oder gar fest in der Microsoft-Unternehmenskultur verankert wurden.

Scrum

Scrum ist unter den heute eingesetzten agilen Methoden vielleicht die bekannteste oder zumindest die mit dem größten Wiedererkennungseffekt, was den Namen betrifft. Im Kern ist Scrum nicht wirklich eine Methode per se, sondern eher ein Gerüst aus Verfahren sowie definierten Rollen für die Beteiligten an seinen Prozessen. Scrum fördert, wie die meisten agilen Methoden, die Bildung kleiner, selbstorganisierender Teams, die an einer klar definierten Reihe von Entwicklungsaufgaben unter Verwendung eines kurzen Releasezyklus arbeiten.

Obwohl Scrum ein nützliches Gerüst für die Bewältigung des Softwareentwicklungsprozesses zur Verfügung stellt, schreibt es kaum spezifische Methoden für die Handhabung der Qualität der zu entwickelnden Software vor. Dies ist aber nicht zwangsläufig problematisch, da Scrum die Verantwortung an die Mitarbeiter innerhalb des Scrum-Teams überträgt, was ihnen die Freiheit gibt und sie darin bestärkt, ihre eigenen qualitativen Verfahren zu implementieren. Andere agile Methoden legen jedoch den Fokus weniger auf Projektgerüste und haben in stärkerem Maße konkrete Vorschriften, welche in spezifischen Verfahren für den Umgang mit der Codequalität zum Einsatz kommen.

Jeder Scrum-Projektzyklus wird durch einen *Sprint* repräsentiert, der eine Zeitspanne bei der Softwareentwicklung darstellt, in der eine festgelegte Reihe an Features fertiggestellt wird. Ein Sprint beginnt mit einer Planungsphase, in der sich die Scrum-Teams dazu verpflichten, den Schwerpunkt ihrer Bemühungen auf die Entwicklung einer festgelegten Reihe an Features zu setzen. Diese festgelegten Features werden vom Team während der Planungsphase aus dem *Product Backlog* ausgewählt, das eine priorisierte Liste aller potenziellen Features der zu entwickelnden Software darstellt. Bis zum Ende der Planungsphase werden alle aus dem Product Backlog ausgewählten Features in das *Sprint Backlog* eingetragen und über dieses verfolgt. Das Sprint Backlog bildet die Details der vom Team durchgeführten Entwicklungsarbeiten an den Features ab. Sobald das Sprint Backlog definiert ist, kann der Sprint beginnen. Üblicherweise dauert ein Sprint 30 Tage. In diesem Zeitraum trifft sich das Team jeden Tag zum *Daily Scrum*, um den aktuellen Status durchzusprechen und sich gegenseitig zu unterstützen, damit jeder im Team produktiv bleibt. Die Vorgabe ist, dass am Ende des Sprints die zuvor im Sprint Backlog definierten Features fertiggestellt und einsatzbereit sind. Abbildung 1.2 veranschaulicht diesen Prozess.

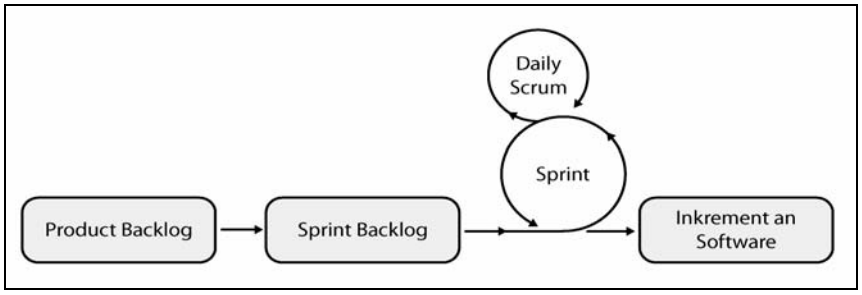

Abbildung 1.2
Prozessfluss von Scrum

HINWEIS Scrum ist ein wirklich interessantes Gerüst, bei dem es sich lohnt, mehr darüber zu lesen. Neben der etwas bunt schillernden Terminologie in Hinblick auf die Rollen der Beteiligten wie *Pigs* (wörtlich: Schweine) und *Chicken* (wörtlich: Hühner) bietet das von Scrum bereitgestellte Gerüst einen hohen Nutzen. Wenn Sie mehr darüber erfahren möchten, ist das Buch »Agile Project Management with Scrum« von Ken Schwaber (Microsoft Press, 2004); deutsche Ausgabe »Agiles Projektmanagement mit Scrum« (Microsoft Press, 2007) sehr empfehlenswert.

Scrum ist vielleicht die am stärksten verbreitete Form agiler Projektmanagementverfahren innerhalb der Entwicklungsteams von Microsoft. Der von Scrum zur Bewältigung der Arbeiten zur Verfügung gestellte Prozess ist tatsächlich ziemlich effektiv, wenn kleine, verhältnismäßig eigenverantwortlich arbeitende Featureteams eingesetzt werden. Die meisten Teams haben Scrum in Verbindung mit unseren Kernentwicklungsverfahren eingesetzt und dabei sehr gute Ergebnisse erzielt.

Extreme Programmierung

Im Unterschied zu Scrum stellt die extreme Programmierung (eXtreme Programming, XP) kein Gerüst aus Verfahren dar, bei denen die Beteiligten interessante Titel tragen und bei dem generell eine interessante Terminologie verwendet wird, die in Verbindung mit Kontaktsport steht.[3] XP kombiniert klar umrissene, nützliche Verfahren mit einer kundenfokussierten Methode zur Lieferung hochqualitativer Software unter Verwendung einer iterativen Herangehensweise.

XP-Befürworter behaupten, dass sich stetig entwickelnde Softwareanforderungen ein natürlicher und wünschenswerter Teil der Softwareentwicklung sind. Anstatt zu versuchen, alle Anforderungen im Vorfeld der Entwicklung zu definieren, fördert XP die Flexibilität in Hinblick auf Anpassungen, um Anforderungen innerhalb des gesamten Projektzyklus hindurch zu ändern. Das Modell stützt sich auf eine sehr enge Arbeitsbeziehung zwischen den Softwareentwicklern und ihren geschäftlichen Ansprechpartnern (den Kunden), wie es Abbildung 1.3 zeigt. Der XP-Entwicklungszyklus stellt eine kontinuierliche Iteration einer Featuredefinition, Schätzung, Featureauswahl und Featureentwicklung dar. Mit jeder Projektiteration werden XP-Teams bei der Erzeugung von solidem Code immer effizienter und effektiver und erreichen eine maximale Wirkung beim Kunden innerhalb eines kurzen Entwicklungszyklus.

[3] Anm. d. Übers: Es wird darauf angespielt, dass in Scrum gewöhnungsbedürftige Titel existieren, wie der SrumMaster oder die schon erwähnten Pigs und Chicken und dass der Ausdruck »Scrum« selbst aus der Kontaktsportart Rugby stammt. Scrum bezeichnet dort einen Mechanismus, um einen Ball nach einem Regelverstoß oder nach einem Aus zurück ins Spiel zu bringen.

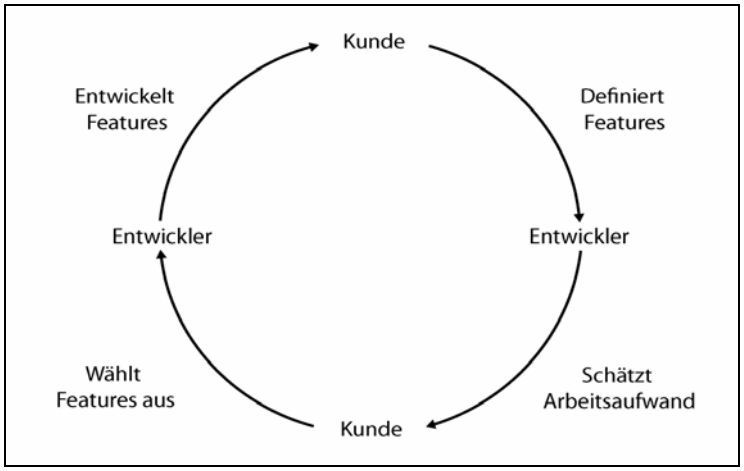

Abbildung 1.3 Prozessmodell bei der extremen Programmierung

Obwohl die Philosophie und die Anwendung von XP in den letzten Jahren Gegenstand von Debatten waren, in denen Mängel von XP wie instabile Anforderungen, das Fehlen von Dokumentation und die mangelnde Skalierbarkeit großer Teams erörtert wurden, sind einige der von XP geförderten Verfahren recht wertvoll, um hohe Codequalität zu erzielen. Zu diesen Methoden[4] gehören beispielsweise

- **Entwicklertests** – Entwickler schreiben durchgehend Code und führen dabei häufig Komponententests (*unit tests*) aus, die bestanden werden müssen, um die Entwicklung fortsetzen zu können
- **Code Refactoring** – Entwickler führen ständig Strukturverbesserungen am System durch, ohne das Verhalten des Codes zu verändern. Der Zweck liegt darin, den Code zu vereinfachen, doppelte Codestellen zu reduzieren und zusätzliche Flexibilität hinzuzufügen.
- **Paarprogrammierung** (*pair programming*) – Entwickler arbeiten zu zweit am Code, um Fehler zu verringern
- **Kontinuierliche Integration** – Features werden jedes Mal, wenn eine Hauptentwicklungsaufgabe fertiggestellt ist, eingebaut und in das Gesamtsystem integriert
- **Programmierstandards** – Es werden Regeln aufgestellt, welche die Kommunikation durch den Code betonen. Diese Kommunikation benötigen die Entwickler, um dem Code folgen zu können

Diese Verfahren sind nicht einzigartig bei XP, aber sie stellen sehr gute Beispiele von Qualitätsverfahren dar, die in vielen Fällen andere agile Methoden übertreffen. Einige dieser Methoden, etwa die Entwicklertests, Code Refactoring und kontinuierliche Integration, werden bereits seit Jahren innerhalb von Microsoft-Entwicklerteams in der Praxis eingesetzt und erweisen sich als recht effektiv. Diese von den Microsoft-Entwicklerteams verwendeten Verfahren werden später in diesem Kapitel noch näher untersucht.

[4] »Extreme Programming Explained: Embrace Change«, Kent Beck und Cynthia Andres (Addison-Wesley Professional, 2004), Seite 54.

Testgetriebene Entwicklung (TDD)

Neben den bereits beschriebenen agilen Methoden sind weitere Methoden wie die testgetriebene Entwicklung *(test-driven development, TDD)* in den letzten Jahren aufgekommen. TDD weist darauf hin, dass Tests die Anforderungen ausdrücken, denen der Code entsprechen muss. Diese Technik erfordert es, dass Entwickler einen automatisierten Komponententest schreiben, bevor sie den Code mit dem Feature in Angriff nehmen. Wenn sich der Featurecode herausbildet, geben die automatisierten Tests unmittelbares Feedback an den Entwickler und melden durch wahr/falsch-Aussagen, ob das Codeverhalten korrekt ist. Da jedes danach entwickelte Feature der Software automatisch einen dazugehörigen Test oder eine Reihe von Tests aufweist, ist häufig ein höheres Vertrauen, was die Qualität des Codes betrifft, die Folge.

Der TDD-Prozess[5] für die Entwicklung von Features schließt die folgenden Schritte ein:

- **Erstellung eines Tests** – Entwickler müssen als Erstes einen Test entwickeln, der die jeweilige Anforderung des Features abbildet
- **Start aller Tests, der neue Test schlägt fehl** – Unter der Annahme, dass eine Reihe von Tests quer durch die Codebasis (den Gesamtquellcode) existiert, sollte der neue Test zwangsläufig fehlschlagen infolge der Ermangelung des neuen Featurecodes.
- **Schreiben des Codes, der dazu führt, dass der Test bestanden wird** – Der Featurecode wird entwickelt, um die Anforderungen des neuen Tests zu erfüllen
- **Start aller Tests und sicherstellen, dass alle erfolgreich sind** – Sobald der neue Featurecode entwickelt wurde, sollten alle Tests erneut ausgeführt werden, um sicherzustellen, dass diese neuen Tests bestanden werden
- **Code Refactoring durchführen** – Der Code, der geschrieben wurde, um die vorangegangenen Schritte zu erfüllen, ist in der Praxis häufig nicht optimiert und sollte nachträglich einem Refactoring unterzogen werden, um doppelt vorhandenen Code oder andere entbehrliche Codeauswüchse zu entfernen.
- **Rinse and repeat (spülen und wiederholen)**[6] – Fortsetzung der Arbeiten, bis keine weiteren zu implementierenden Anforderungen mehr vorhanden sind

Andere agile Methoden fördern Komponententests, die vom Entwickler erstellt werden, der die Features realisiert hat, aber TDD steht hier abseits mit seiner strengen Auffassung, dass die Tests vor der Featureentwicklung stehen (*testing before coding*). TDD stellt ein interessantes Angebot mit hohem Nutzen in Gestalt eines Prozesses zur Verfügung, der hohe Codequalität sicherstellt; jedoch bietet es kein durchgehendes Gerüst zur Verwaltung des Softwareentwicklungszyklus an. Das Ziel von TDD liegt einfach darin, ein Gerüst bereitzustellen, mit dem Kundenanforderungen über einen iterativen Ansatz in Hinblick auf die Tests und die Programmierung angegangen werden.

---

[5] »Test-Driven Development in Microsoft .NET«, James W. Newkirk und Alexei A. Vorontsov (Microsoft Press, 2004).

[6] Anm. d. Übers: In Anlehnung an Haarshampoos, auf denen oft solche Bedienungshinweise zu finden sind, die ein Ausspülen und eine Wiederholung der Anwendung empfehlen. »Rinse and repeat« ist im englischsprachigen Raum zu einer Redewendung geworden, die Übersetzung »Spülen und wiederholen« ist im Deutschen als Redewendung dagegen kaum bekannt. Bezogen auf das Softwareverfahren ist hier nur gemeint, dass die einzelnen Schritte zu wiederholen sind.

Agile Verfahren in der Entwicklungskultur von Microsoft

Es ist kein Geheimnis, dass Microsoft das größte Softwareunternehmen der Welt ist. Das Unternehmen hat sich dieses Prädikat nicht durch die Anzahl seiner Angestellten alleine verdient, sondern vielmehr durch die unvorstellbare große Anzahl an Produkten, die jedes Jahr von Microsoft-Entwicklern sowie Entwicklungsteams quer durch die vielen Kontinente, auf denen das Unternehmen eine Entwicklungspräsenz unterhält, erstellt werden.

Für seine Größe ist Microsoft ein unglaublich bewegliches Unternehmen, und die Entwicklungsteams sind allgemein sehr agil, was ihre Herangehensweise betrifft, Softwareprodukte auf den Markt zu bringen. Beispielsweise sind die Teams in der *Online Services Division* von Microsoft, etwa *Windows Live*, bekannt dafür, dass sie in der Lage sind, mehrmals im Jahr große Updates mit vielen neuen Features an einen weltweiten Kreis von mehreren Hundert Millionen Benutzern auszuliefern. Betrachten Sie die Größe und Komplexität von Produkten wie Microsoft Windows oder Microsoft Office. Diese Produkte benötigen sehr große, gut koordinierte und unvorstellbar präzise Entwicklungsprozesse, die in hohem Maße auf die Codequalität fokussiert sind. Trotz der Größe und Komplexität dieser Produkte, der Codebasis und der Entwicklungsteams kann Microsoft weiterhin die Auslieferung neuer Versionen dieser Produkte in einem ordentlichen Intervall leisten (Office-Versionen etwa erscheinen im Durchschnitt alle zwei Jahre). Wie Microsoft-Teams dies zustande bringen, ist auf jahrelange kontinuierliche Weiterentwicklung und Verbesserung der Entwicklungsverfahren und automatischen Tools zurückzuführen, bei der jedes an sich ein komplettes Buch füllen könnte. Jedoch gibt es einige der oben genannten agilen Verfahren, deren Erwähnung sich lohnt.

HINWEIS Ein tiefer gehendes, sehr empfehlenswertes Beispiel, wie Microsoft-Teams die Entwicklungsverfahren kontinuierlich weiterentwickelt haben, finden Sie im Windows Engineering Team-Blog unter *http://blogs.msdn.com/e7/* (engl.).

Viele Jahre lang, vielleicht ein Jahrzehnt oder länger, haben sich die Microsoft-Entwicklungsteams verschiedene *agile* Verfahren einverleibt, um die Qualität zu verbessern. Insbesondere integrierten die meisten Teams optimale Programmierverfahren, von Entwicklern selbst erstellte Komponententests, Refactoring und kontinuierliche Integration in ihre tägliche Entwicklungspraxis. Teams wie das Office-Team erstellen und verteilen ihren aktuellen Arbeitsbuild auf täglicher Basis. Der Code von unterschiedlichen Entwicklungsteams wird laufend integriert und getestet, wobei hier optimal abgestimmte Erstellungs- und Testsysteme zum Einsatz kommen. Das Ergebnis ist eine tägliche Version des kompletten Produkts von zuverlässiger Qualität. Wie Sie sich vielleicht vorstellen können, erfordern diese Prozesse eine recht hohe technische Investition, aber das Resultat ist ein *angekurbelter* Prozess, der auf einen reproduzierbaren und hochqualitativen Output optimiert ist. Microsoft Visual Studio bietet eine sehr gute Reihe an Tools an, die es kleinen wie auch großen Entwicklungsteams in vergleichbarer Weise ermöglichen, sowohl ihre Buildprozesse unter Verwendung von MSBuild als auch Testabdeckungen mithilfe der Testtools aus Visual Studio Team System (VSTS) zu automatisieren. MSBuild und die VSTS-Testtools stellen Funktionen für tägliche Builds sowie Zusammenstellungen automatisierter Tests in unbeaufsichtigten Umgebungen zur Verfügung.

Vorziehen der Qualitätsverfahren

Während jeder Phase des Entwicklungszyklus leiten die Qualitätsverfahren die Programmmanager, Entwickler und Tester in Richtung des besten Endprodukts an, das sie gemeinsam herstellen können:

- Am Anfang des Entwicklungszyklus liegt der Schwerpunkt auf der Erfassung von Erkenntnissen und Feedback des Kunden, der Ausführung der detaillierten Planung, auf der Featurespezifikation sowie dem übergreifenden Systemdesign
- Während der Implementierungsphase des Entwicklungszyklus setzen die Entwickler den Schwerpunkt auf die Anwendung der Qualitätstechniken auf den von ihnen zu entwickelnden Code, indem sie Standards, gemeinsame Codeverwendung und Codeoptimierung sowie Komponententests einsetzen
- Vor dem Release führen Tester breite System- und Integrationstests durch, um sicherzustellen, dass das Produkt durchgehend von hoher Qualität ist

Dieser Blickwinkel impliziert, dass die Integration der Qualitätsverfahren alle Phasen des SDLC durchdringen sollte. Dies ist nicht immer der Fall, jedoch ist anzumerken:

- Die Wasserfallmethode fördert die Sicherstellung der Qualität durch eine auf das Projektvorfeld fixierte Planung sowie ein breites Spektrum an Tests nach der Programmierung
- Agile Methoden setzen den qualitativen Hebel inmitten des laufenden Entwicklungszyklus an und stützen sich auf spezifische Verfahren, um zu gewährleisten, dass die Codequalität während des gesamten Projekts hindurch hoch bleibt.

Bei der Wasserfallmethode schreiben Entwickler Code und geben ihn an Tester weiter, die den Code auf Fehler untersuchen. Bei diesem Modell, bei dem die Features und die Komplexität von den Entwicklern in einem hohen Tempo hinzugefügt werden, wachsen Umfang, Aufwand und Ballast beim Test der gesamten Anwendung signifikant an. Diese Anhäufung von Testarbeiten in einer späten Phase des Projektzyklus erhöht das Risiko einer Qualitätsminderung der Software und kann auf die Entwicklung neuer Features und die langfristige Wartbarkeit des Systems einen lähmenden Effekt nach sich ziehen. Einfach ausgedrückt: Je mehr Code geschrieben wird, desto schwieriger gestaltet sich der Test der Software.

HINWEIS Im Buch »Debugging Microsoft .NET 2.0 Applications« von John Robbins (Microsoft Press, 2006); deutsche Ausgabe »Microsoft .NET 2.0 Anwendungen debuggen: Praktische Informationen zum Debuggen mit Visual Studio 2005« (Microsoft Press, 2006) behauptet der Autor, dass der Featurewildwuchs möglicherweise den größten Beitrag daran hat, wenn die Qualität einer Software sinkt. Es dürfen dann keine Fehler mehr unterlaufen, denn das Hinzufügen von Features in einer späten Phase des Entwicklungszyklus ist eine Rezeptur für Schwierigkeiten. Es kann jedoch Umstände geben, unter denen Sie so stark dem Projekt- oder geschäftlichem Druck ausgesetzt sind, dass Ihnen kaum etwas anderes übrig bleibt, als diese späten Erweiterungen zu akzeptieren. Es ist wichtig, für alle Features strenge Qualitätsprozesse einzurichten, unabhängig davon, für welchen Liefertermin Sie sich verpflichtet haben.

Verschiebungen bei den Qualitätsverfahren wie vorgezogene oder noch früher im Entwicklungszyklus stattfindende Tests, wie sie agile Methoden, etwa XP, fördern, reduzieren die Risiken der Veröffentlichung von fehlerbehafteter Software. Außerdem lässt sich durch die Sicherstellung, dass jedes Feature zunächst verifiziert und getestet wird, bevor es über die Versionsverwaltung eingecheckt wird, erreichen, dass die Agilität des Entwicklungsteams ansteigt. Die Gründe dafür liegen in der geringeren Anzahl gescheiterter Builds, in der Entstehung von weniger Bugs sowie in einem Featureentwicklungsprozess, der in höherem Maße reproduzierbar ist und eine höhere Stabilität aufweist. Grundsätzlich sollte die Entwicklung und

Programmierung von Features nur so schnell erfolgen, wie die dazugehörigen Tests mithalten können. Dies erlaubt es, dass die Softwareentwicklung mit einem einheitlicheren und überschaubaren Tempo voranschreitet und einen qualitativ höherwertigeren Output erzielt. Dies ist einleuchtend, wenn man sich vergegenwärtigt, wie agile Methoden – z.B. XP – die inhärente Testbarkeit des Codes fördern und Investitionen in automatisierte Testframeworks wie das von VSTS oder NUnit empfehlen, um zu erreichen, dass bei der Entwicklung kontinuierlich automatisierte Testreihen zur Ausführung gelangen, die den Code entsprechenden Prüfungen unterziehen.

Die Verlagerung von Qualitätsverfahren im Entwicklungszyklus nach vorne begünstigt die frühe Aufdeckung von Fehlern im Code, was es den Softwareentwicklern letztlich erleichtert, auf diese Probleme angemessen zu reagieren, ihre Designs falls nötig zu korrigieren und die sich aus dem Zeitplan ergebenden Verpflichtungen einzuhalten. Die Verantwortlichkeit für die Qualität der Software wird gemeinsam an die Entwickler und Tester verteilt, und der SDLC wird agiler und besser vorhersehbar. Diese Philosophie ermuntert nicht nur zur strategischen Anwendung besserer Qualitätsverfahren, sondern fördert auch eine Qualitätskultur innerhalb der Entwicklerteams. Diese Verbindung von Qualität und Agilität wird erreicht, indem sehr gute Entwicklungsverfahren mit einer Qualitätskultur sowie einer Auslieferungspolitik – in Bezug auf Strenge und Intervall – kombiniert werden, die den Anforderungen Ihres Unternehmens am besten entsprechen.

Einblick in Microsoft: Entwicklung von Windows Live Hotmail

Windows Live Hotmail ist ein webbasiertes E-Mail-Produkt, das von Microsoft innerhalb der *Windows Live Suite of Services* angeboten wird. Hotmail stellt vielleicht einen der bekanntesten aller Internet-E-Mail-Dienste dar und war bei seinem Start 1996 einer der ersten freien, webbasierten E-Mail-Dienste. Viele Jahre lang wurde Hotmail unter dem Label der MSN-Dienste angeboten, bevor es in Gestalt von verwaltetem Code nachgebaut und 2007 unter dem Label Windows Live neu gestartet wurde. Derzeit ist Hotmail das weltweit meistverwendete webbasierte E-Mail-Produkt und wird von mehr als weltweit 250 Millionen Benutzern[7] in mehr als 35 Sprachen genutzt. Die Hauptkomponenten der Anwendung umfassen webbasierte E-Mail, Kalender, Kontaktverwaltung sowie Instant Messaging-Features.

Seitdem Hotmail von Microsoft 1997 übernommen wurde, hat es eine vollständige Verwandlung durchgemacht, von einer Unix-basierenden Anwendung hin zur heutigen, auf verwaltetem (*managed*) Code basierenden Anwendung, die unter Verwendung von Windows Server-Technologien läuft. Es stellt eine große, hochentwickelte und in mehrere Komponenten unterteilte Anwendung dar. Bei den Komponenten handelt es sich unter anderem um Subsysteme zur Speicherung, für den E-Mail-Versand und das Web-Frontend. In der verteilten (also ausgelieferten) Version umfasst alleine die Größe des Web-Frontend-Subsystems mehr als 250 MByte. Dies ist ein Indikator für die Größe von Hotmail als Anwendung, repräsentiert aber in keinster Weise die Komplexität des Systems.

[7] comScore, Inc., Pressemitteilung vom 5. Februar 2008, *http://www.comscore.com/press/release.asp?press=2196* (engl).

Da Hotmail primär eine E-Mail-Anwendung darstellt, muss es extrem skalierbar, zuverlässig und sicher sein und größtmöglichen Schutz vor internetbasiertem Missbrauch durch Spam, Viren und Phishing bieten. Die Benutzer erwarten von Personal Information Management-Anwendungen wie Hotmail zu Recht ein hohes Maß an Qualität und dementsprechend ein konkurrenzfähiges Produkt mit erstklassigen E-Mail-, Kalender- und Kontaktverwaltungsfunktionen, das knallhart gegenüber Wettbewerbern auf dem Markt aufgestellt ist. Um seine Position als globaler Marktführer in der webbasierten Kommunikation zu behaupten, muss der Entwicklungsprozess von Hotmail rechtzeitige und wiederholbare Lieferungen von Anwendungscode und Infrastruktur bei hoher Qualität, Zuverlässigkeit und Skalierbarkeit gewährleisten.

Entwicklungsprinzipien

Das Windows Live Hotmail-Entwicklungsteam hat im Lauf der Jahre bei der Lieferung hochqualitativer, zuverlässiger und funktionsreicher Webanwendungen an eine globale Zielgruppe große Erfahrung gesammelt und sehr viel Sachkenntnis bewiesen. Als Team erkennen die Mitarbeiter die Tiefe und Komplexität der Herausforderungen, denen sie bei der Entwicklung begegnen, und sie haben Jahre damit verbracht, ihre Verfahren zu perfektionieren. Sie halten an einer Reihe von Grundprinzipien fest, welche die Art und Weise steuern, mit der Software auf den Markt gebracht wird. Diese Prinzipien umfassen folgende:

- **Fokus auf die Qualität des Dienstes, auf Performance und Sicherheit setzen** – Dies ist das vielleicht wichtigste aller Merkmale, sodass das Team dieses Prinzip über alle anderen stellt. Die Kultur des Teams fördert ein hohes Maß an Produktqualität, und das Team glaubt daran, dass Hotmail ein zuverlässiger, schneller und sicherer Dienst für seine Benutzer ist. Dieser Glaube durchdringt die Entwicklungsverfahren des Teams beginnend mit dem Design, der Programmierung, den Tests hin bis zur Websitearchitektur und -infrastruktur. Obwohl viele Unternehmen eine ähnliche Kultur anstreben, hat das Hotmail-Team bereits eine solche erreicht, indem es den qualitativen Schwerpunkt auf alle Aspekte seines Entwicklungsprozesses gesetzt hat.
- **Wirksamer Einsatz iterativer Entwicklung** – Sämtliche Disziplinen innerhalb des Teams (die Programmmanagement, Entwicklung und Test einschließen) arbeiten zusammen, um die Software unter Verwendung iterativer Entwicklungsmethoden zu definieren, zu entwickeln und zu testen. Das Hotmail-Team hat aus der eigenen Erfahrung und auch aus Erfahrung anderer Microsoft-Teams gelernt, dass iterative Entwicklung zu einer Qualitätsverbesserung des Codes und zu einer Verringerung der Anzahl an Bugs während des gesamten Entwicklungszyklus hindurch beiträgt, während gleichzeitig die Effizienz des Teams ansteigt. Das Team wendet verschiedene Strategien an, um eine erfolgreiche iterative Entwicklung sicherzustellen. Dabei trägt es vor allem dafür Sorge, dass Iterationen kurz bleiben (sechs bis acht Wochen), eine durchgehende Fertigstellung von Features erfolgt, bevor mit neuen Features begonnen wird, die Arbeit in kleinen und selbstständig agierenden Teams stattfindet und ein frühzeitiges und häufiges Kommunizieren von Informationen in Hinblick auf den Status seines Projekts geschieht.
- **Voraussagbare und wiederholbare Prozesse sicherstellen** – Im Lauf der Jahre war das Hotmail-Team bei der Lieferung von Produktqualität meist erfolgreich, da das Team sich auf voraussagbare und wiederholbare Prozesse stützen konnte. Beispielsweise halfen gebräuchliche Webserverhardware, Softwareverteilungsmethoden und Verwaltungstools dem Team, hohe Effizienz bei der Verwaltung seiner Webinfrastruktur zu erlangen. Planungs- und Entwicklungsprozesse unterstützten das Team dagegen bei der Risikominimierung in Hinblick auf Lieferfristen und verbesserten die gesamte Lieferqualität.

- **Einsatz gebräuchlicher Tools, Prozesse und Terminologie** – Wie die meisten Teams bei Microsoft ist das Hotmail-Team Teil einer größeren Organisation von Produktentwicklungsteams innerhalb von Windows Live. Aus Effizienzgründen verwenden diese Teams eine gemeinsame Basis gebräuchlicher Tools und Prozesse wie Versionsverwaltung, Buildprozesse sowie Datenbanken zur Fehlerverfolgung. Die übergreifenden Entwicklungsprozesse und die Terminologie sind ebenso quer über die Teams einheitlich gehalten, was letztendlich sowohl hilft, den Softwareauslieferungsprozess zu rationalisieren, als auch die Gesamtkommunikation zwischen den Teams zu verbessern.

Schlüsselfaktoren für den Erfolg

Seitdem sich das Hotmail-Team vor einigen Jahren von einem Wasserfallmodell wegbewegt hat, stellte es fest, dass die oben genannte Reihe an Prinzipien einen sehr großen Effekt auf die Codequalität und die Gesamtagilität des Teams hat. Diese Prinzipien halfen dem Team gemeinsam, weiterhin Neuerungen an seinen Dienstleistungen einzubringen, während sie weiterhin einen hohen Grad an Qualität und Zuverlässigkeit erreichten. Jedoch erwähnte das Team seine Bewegung hin zur iterativen Entwicklung und bezeichnete im Besonderen die spezifischen Verfahren, bei denen qualitative Aspekte vorgezogen werden, als die wirkungsvollsten in seinem Softwareprozess. Im engeren Sinn empfand das Team, dass die Verfahren, die den größten positiven Effekt auf die Qualität seiner Arbeit während der iterativen Entwicklung hatten, folgende einschließen:

- **Auslieferung testfähiger Komponenten** – Dieses Verfahren stellt sicher, dass die Entwicklung von Features nur so schnell fortgesetzt wird, wie die Features überprüft und getestet werden können. Sobald ein Feature programmiert ist und für den Test bereitsteht sowie den erforderlichen, vom Entwickler geschriebenen Komponententest bestanden hat, wird es für einen gründlicheren Testdurchgang herausgegeben, bevor in den Arbeitsast des Quellcodebaums eingecheckt wird. Dieser Prozess gewährleistet, dass das entwickelte Feature von ausreichend hoher Qualität ist, um in den Arbeitsast des Quellcodebaums aufgenommen zu werden und voraussichtlich kein Scheitern eines Builds des Gesamtsystems nach sich ziehen wird.
- **Tägliche Builds** – Während die Entwicklung von Features voranschreitet und testfähige Komponenten in den Arbeitsast des Quellcodes integriert werden, wird jeden Tag automatisch ein Build des Gesamtsystems erstellt. Die täglichen Builds werden von einer Reihe automatisierter Tests begleitet, die als *build verification tests* (BVTs) bezeichnet werden und die helfen, sicherzustellen, dass keine Eincheckvorgänge vom Vortag neue Bugs in die Kernfeatures der Anwendung eingeschleust haben. Dieser Prozess unterstützt das Team bei der Bewerkstelligung und Aufrechterhaltung der Gesamtqualität der Anwendung auf einer täglichen Basis. Fehlgeschlagene Builds werden sehr ernst genommen und im Falle von Bugs, welche von den BVTs erkannt werden, wird sämtliche Arbeit an den Features so lange unterbrochen, bis das Problem gelöst und die Testreihe bestanden wird.

- **Automatisierte BVTs zusammen mit den täglichen Builds durchführen** – Diese Reihe an automatisierten Tests stellt sicher, dass die Grundfunktionalität der Anwendung von hoher Qualität ist und dass der Build für einen stärker formalen, durchgehenden Test freigegeben werden kann. Die Automatisierung gewährleistet, dass die Tests wiederholbar sind und schnell ausgeführt werden können, um die Qualität der Builds zu untersuchen. Die Ausführung und das erfolgreiche Bestehen dieser Tests auf einer täglichen Basis stellen sicher, dass die Qualitätslatte während des gesamten Entwicklungszyklus gleich bleibend hoch hängt. Mindestens einmal in der Woche gibt das Team einen dieser täglichen Builds an eine interne *dogfood*-Umgebung (dazu gleich mehr), um es Microsoft-intern frühzeitigen Benutzern (*early adopters*) zu erlauben, den aktuellsten Produktbuild zu verwenden und dem Team Feedback zu geben.
- **Frühe und häufige Auslieferung an eine *dogfood*-Umgebung** – *dogfood* (wörtlich übersetzt *Hundefutter*) – in der längeren Version *eating your own dogfood* (*das eigene Hundefutter essen*) – ist ein umgangssprachlicher Ausdruck bei Microsoft, der aussagt, dass man bereits in einer frühen Entwicklungsphase Vertrauen zu seinem eigenen Produkt hat.[8] Das Hotmail-Team glaubt an den Gedanken der frühen und häufigen Auslieferung an seine Benutzer und stellt, wie bereits erwähnt, eine wöchentliche Vorabversion des Produkts einer kleinen Gruppe von Microsoft-Angestellten zur Verfügung, die erwartungsvoll eine neue Version des Produkts bewerten und testen möchten. Das Team ist der Überzeugung, dass ihm ein Haften an einem iterativen Rhythmus von Programmierung, Build, Test und Veröffentlichung an eine *dogfood*-Umgebung hilft, die durchgehende Qualität des Produkts so hoch wie möglich durch den gesamten Entwicklungszyklus hindurch aufrechtzuerhalten.

Während der Entwicklung von Windows Live Hotmail, das 2007 veröffentlicht wurde, bewegte sich das Team hin zu einem stärker iterativen Entwicklungsmodell, da es damit beschäftigt war, die gesamte Anwendung in verwaltetem Code nachzubauen. Durch eine wirksame Anwendung der oben erwähnten Verfahren, etwa die Herstellung testfähiger Komponenten, tägliche Builds und die frühe sowie häufige Veröffentlichung an eine *dogfood*-Umgebung, erkannte das Team, dass die Qualität seiner Arbeit im Vergleich zu den Erfahrungen unter Zuhilfenahme eines Wasserfallmodells drastisch anstieg. Insbesondere bemerkte das Team, dass die Buganzahl beim Einsatz einer iterativen Entwicklungsmethode langsamer anstieg und in einer viel geringeren Anzahl als bei der Wasserfallmethode ihren Höhepunkt erreichte. Die Wasserfallmethode führte oft dazu, dass das Team am Ende des Entwicklungszyklus mit einer hohen Anzahl an Bugs konfrontiert wurde. Das langsamere Bugwachstum und die geringere Gesamtanzahl an Bugs erlaubte es dem Team, die Bugs schneller zu beheben und die Kontrolle über die Codeänderungen aufrechtzuerhalten, was die kontinuierliche Stabilität des Produkts sicherstellte. Eine Gegenüberstellung der Buganzahl unter Verwendung eines Wasserfallentwicklungsmodells mit einem iterativen Entwicklungsmodell wird in Abbildung 1.4 illustriert.

[8] Anm. d. Übers: Der Ausdruck »eating your own dogfood« stammt von einem TV-Werbespot für ein Hundefutter, von dem der Darsteller behauptet, es wäre so gut, dass er seine eigene Hunde damit füttert. (Quelle: *http://en.wikipedia.org/wiki/Eat_one%27s_own_dog_food*)

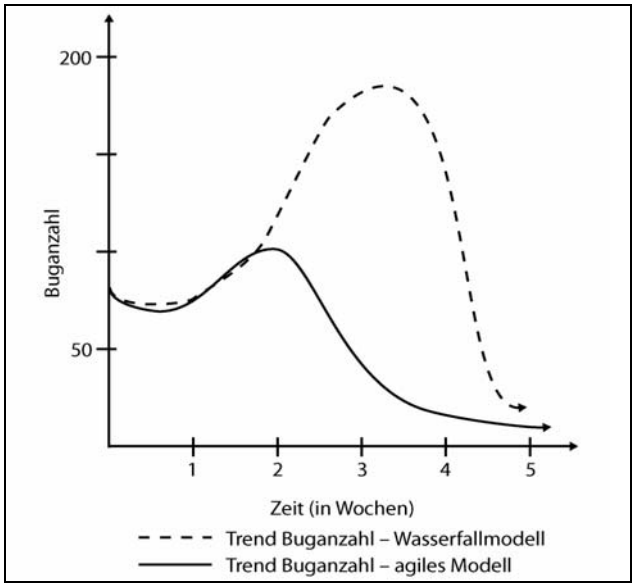

Abbildung 1.4 Vergleich der Bugtrends bei der Entwicklung von Windows Live Hotmail

Das Team schreibt seinen Erfolg mit der Veröffentlichung eines hochqualitativen Produkts den sich wiederholenden Prozessen zu, die es zur Steuerung der Anwendungsqualität in frühen Phasen des Entwicklungszyklus eingerichtet hat. Das Hotmail-Team hat mit seiner Erfahrung bewiesen, dass die Implementierung qualitätsfokussierter Verfahren – etwa der Auslieferung testfähiger Komponenten, der Durchführung täglicher Builds und der Verwendung automatischer Testreihen – in einem vorgezogenen Entwicklungszyklus sowohl ein höheres Maß an Qualität in der Anwendung als auch ein besser vorhersehbares Ergebnis des Entwicklungszyklus sicherstellt. Im Februar 2007 wurde dies bestätigt, als Hotmail die Auszeichnung *Editor's Choice* des PC Magazine für webbasierte E-Mail-Produkte erhielt. Diese Auszeichnung ist eindeutig ein Beleg dafür, dass es dem Team gelungen ist, den Fokus auf die Lieferung von Anwendungscode und Infrastruktur mit hoher Qualität, Zuverlässigkeit und Skalierbarkeit zu legen.

Strategien für das Schreiben von solidem Code

Das Prinzip des Vorziehens qualitativer Aspekte verdeutlicht die Wichtigkeit der Implementierung qualitätsfokussierter Entwicklungsverfahren in die täglichen Arbeitsabläufe und deren Integrationsstrenge. Werden keine Qualitätsverfahren, wie früh innerhalb des Entwicklungszyklus stattfindende Tests angewendet, staut sich wahrscheinlich sehr viel Testarbeit bei jedem zusätzlich hinzugefügten Feature an. Anschließend wird das Risiko erhöht, dass es zu einer Überziehung des Liefertermins der Software kommt und dass die Software von geringer Qualität ist. Für viele Unternehmen bedeutet dies neben einer strategischen auch eine kulturelle Änderung, aber das Sich-zu-Eigen-Machen dieser Kultur und dieser Verfahren führt letztlich zu einer besseren Softwareentwicklung und der Auslieferung von solidem Code. Im weiteren Verlauf dieses Buchs werden sowohl strategische als auch kulturelle Empfehlungen für die Verbesserung der Gesamtqualität der Software gemacht, die Sie und Ihr Team erzeugen. Der Rest dieses Kapitels beschreibt diese Empfehlungen kurz in Gestalt einer thematischen Gliederung (z.B. *Fokus auf dem Design*). Die Unterabschnitte (z.B. *Klassendesign und Prototyping*) jedes Themas bilden dabei die verbleibenden Kapitel des Buchs.

Fokus auf dem Design

Das Design ist die vielleicht erste Verteidigungslinie gegen die Einführung mangelhafter Qualität in Ihre Softwareanwendung. Herkömmliche und agile Methoden stimmen darin überein, dass das Design ein kritischer Schritt im Prozess bei der Herstellung hochqualitativer Software ist. Es gibt zahlreiche ausgezeichnete Designprinzipien, die ganze Bücher füllen, die von unvorstellbar klugen Menschen geschrieben sind. Wie Sie sich vielleicht vorstellen können, war es recht schwierig, eine kleine Reihe von Designprinzipien auszuwählen, auf die in diesem Buch der Schwerpunkt gelegt wird. Glücklicherweise habe ich die letzten paar Jahre meiner Laufbahn mit der Arbeit an Softwareanwendungen unterschiedlicher Größe verbracht, von denen viele in verwaltetem Code entwickelt wurden. Die nahe liegende Wahl bestand darin, die Designprinzipien zu fokussieren, die sich meiner Erfahrung nach als die nützlichsten erwiesen haben.

- **Klassendesign und Prototyping** – Wenn eine Anwendung entwickelt wird, wird bei der objektorientierten Programmierung mit dem Klassendesign und dem Prototyping begonnen. Im Allgemeinen fängt dies mit der Bestimmung der *Beteiligten* an, das heißt der *Typen* der Anwendung, und wie diese untereinander in Beziehung stehen und interagieren. Das Interaktions- und Beziehungsmodell wird in Designdokumente übersetzt, wobei hierfür die Unified Modeling Language (UML) zum Einsatz kommt. Von da aus wird ein Prototyp erstellt und die Konzepte, Beziehungen und Interaktionen werden auf Korrektheit und Vollständigkeit getestet. Während dieser Phase ist es am wichtigsten, die Risiken zu analysieren, die mit dem angestrebten Design verbunden sind, bevor die (eigentliche) Implementierung startet.
- **Metaprogrammierung** – Die Reduzierung der Komplexität und die Verbesserung der Wartbarkeit einer Software tragen direkt zur qualitativen Steigerung der Anwendung bei. Durch eine Abstraktion von Verhaltensweisen außerhalb des Codes und innerhalb der Anwendungsmetadaten wird das Anwendungsdesign von Natur aus robuster, flexibler und anpassungsfähiger. Dies führt dazu, dass Anwendungs- und Testcode erheblich stärker vereinfacht werden können. In Anwendungen in verwaltetem Code wird der Einsatz von XML als Metaprogrammiersprache als Mittel unterstützt, um das Anwendungsverhalten zur Laufzeit zu modifizieren. Es handelt sich um das Designverfahren, das letztlich zur Kostenreduzierung bei der Wartung und zu höherer Flexibilität führt, insbesondere in Situationen, wenn Erweiterungen und Änderungen an bestehenden, bereits in Betrieb befindlichen Anwendungen vorgenommen werden sollen.
- **Performance** – Eine gute Performance einer Anwendung kann als eigenes Feature gesehen werden, da es die Wettbewerbsfähigkeit einer Software stärkt. Die Qualität einer Anwendung ergibt sich nicht nur aus der Frage, ob sie leicht sichtbare Bugs aufweist, sondern wird auch durch die Gesamterfahrung beeinflusst, die Endbenutzer beim Einsatz der Anwendung machen. Letzteres ist insbesondere dann wichtig, wenn Livedienste über das Web oder intelligente Clients hergestellt werden, die sich auf internetbasierte Endpunkte stützen. Denn bei Onlineanwendungen können Netzwerklatenz, Nutzdatengröße und sogar bestimmte Protokolle zu negativen Eindrücken beim Endbenutzer führen. Als einer der Hauptgrundsätze bei der Entwicklung sollten Performancebetrachtungen Teil jedes Designs sein. Eine Verschiebung dieser Überlegungen in eine späte Phase des Entwicklungszyklus hinein kann signifikante Codeumgestaltungen nach sich ziehen, wenn Performancemängel entdeckt werden.
- **Skalierbarkeit** – Wenn Software sich von einer reinen Desktopanwendung hin zu einem serverbasierten Modell bewegt, wird die Skalierbarkeit zu einem kritischen Element in hochqualitativen Softwaredesigns. Es liegt in der Natur serverbasierter Anwendungen, dass die Verwaltung von Ressourcen

direkten Einfluss auf die Performance hat und auch auf die mit der Skalierung Ihrer Anwendung verbundenen Kosten, um die Benutzeransprüche zu erfüllen. Skalierungsfaktoren sollten immer als ein Teil des Softwaredesigns betrachtet werden, und in vielen Fällen erzwingen sie sogar Kompromisse bei bestimmten Features. Das Hinauszögern der Skalierungsbetrachtungen jenseits der Design- und Programmierungsphase des Entwicklungszyklus schraubt nicht nur die Betriebskosten hoch, sondern kann auch Designänderungen sowie nachträgliche größere Codeänderungen und -erweiterungen nach sich ziehen, wenn Bugs und allgemeine Probleme mit der Anwendung erst unter einer größeren Last entdeckt werden.

- **Sicherheit** – Der vielleicht wichtigste der hauptsächlichen Grundsätze und häufig die Quelle für viele Qualitätsprobleme ist der Aspekt der Sicherheit, der für alle Entwickler ein Hauptschwerpunkt beim Design sein sollte. Als Software und Benutzer stärker vernetzt wurden, haben die Angriffe auf Software signifikant in Frequenz und Schärfe zugenommen. Das Design muss daher eine sorgfältige Betrachtung von Angriffen auf die Software und entsprechende Gegenmaßnahmen in Gestalt von Bedrohungsmodellierung (*threat modeling*) einschließen. Die Aufnahme der Sicherheitsaspekte in den gesamten Entwicklungszyklus und insbesondere in das Design führt letztendlich zu mehr Sicherheit und einem besseren Schutz für die Benutzer Ihrer Software.

Fehlerabwehr und Debugging

So wie Design ein kritisches Element bei der Entwicklung von solidem Code darstellt, genauso ist es wichtig, die Fallstricke zu kennen, die einen an der Auslieferung fehlerfreier Software hindern. Das Erfassen typischer Programmierfehler kann weitestgehend oder vollständig bekannte potenzielle Probleme verhindern. Darüber hinaus müssen Entwickler bei der Codeerstellung defensive Strategien einsetzen, die ihnen dabei helfen, weniger augenscheinlichere Arten von logischen Bugs zu vermeiden. Reicht ein proaktiver Abwehrcode nicht aus, unterstützt ein aktiv testender Code effektiv die Beseitigung entsprechender Probleme. Zum Schluss sollten jegliche verbliebene Bugs mithilfe eines aggressiven Debuggings verfolgt werden. Diese Prinzipien sind für alle Entwickler geeignet, die in verwaltetem Code programmieren und die qualitative Probleme in ihrem Code proaktiv angehen möchten.

- **Speichermanagement** – Das Speichermanagement im verwalteten Code unterscheidet sich erheblich von dem im nativen Code. Jedoch ist der Unterschied nicht nur technischer Natur, er ist auch philosophisch. Bei nativem Code sind die Entwickler im Wesentlichen für den Zustand des Arbeitsspeichers verantwortlich, er ist gewissermaßen ihr Eigentum. Dies ändert sich bei verwaltetem Code. Der Entwickler muss hier die Verantwortung mit der Common Language Runtime (CLR) teilen – insbesondere dem Garbage Collector. Dies erfordert es, dass der Entwickler in einigen Fällen den Garbage Collector beauftragen muss, etwa eine Speicherbereinigung auszuführen.
- **Defensive Programmiertechniken** – Die Anwendung defensiver Techniken bei der Softwareentwicklung hilft proaktiv, die Anzahl in die Anwendung eingeschleuster Bugs zu begrenzen. Diese Techniken können so einfach sein wie die Durchführung häufiger Codeüberprüfungen und das Setzen von Compileroptionen oder fortschrittlicher, wie die Verwendung musterbasierter Programmierung, um zuverlässige und getestete Algorithmen wirksam einzusetzen. Ein Einsatz dieser Verfahren in Ihrem Softwareentwicklungsprozess kann im hohen Maße das Potenzial für logische Bugs reduzieren.

- **Debugging** – So wie die Elemente des Softwareentwicklungszyklus iterativ sind, so ist es auch der Prozess des Debuggens. Die meisten Entwickler assoziieren Debugging mit Bugs, die nach der Freigabe der Software auftauchen, oder mit Laufzeitproblemen. Jedoch verbessert proaktiver Debugcode, der sicherstellt, dass das Codeverhalten unter allen Umständen wie erwartet ausfällt, die Releasequalität ganz erheblich. Wenn sich Bugs im Build oder im fertigen Release bemerkbar machen, sollte jedoch ein Softwareentwickler, der mit einem Debugging-Tool wie dem von Visual Studio 2008 vertraut ist, die Probleme lösen können.

Analyse und Test

Neben der Implementierung durchdachter Designs und der Anwendung bewährter Entwicklungsverfahren muss der Code »beschäftigt«, also in Anspruch genommen werden. Dies ist sehr wichtig, um zu verstehen, wie sich die Anwendung verhält, wenn die Benutzer beginnen, mit ihr zu interagieren. Wohingegen manuelles Testen eine sehr gute Methode für die Aufdeckung von Mängeln in Hinblick auf die Benutzerfreundlichkeit von Anwendungen darstellt, dienen programmgesteuerte Komponententests und die Testabdeckung (*code coverage testing*) fast immer dazu, Bugs früh aufzuspüren, und helfen, die Qualität des Gesamtcodes zu verbessern. Diese Prozesse sollten in den Entwicklungszyklus integriert werden, sodass die Codequalität in Echtzeit während der Featureentwicklung angegangen werden kann und nicht erst, nachdem die Entwicklung beendet ist.

- **Codeanalyse, -abdeckung und -test** – Sobald Codeanalyse, Komponententests und Testabdeckung innerhalb der Softwareentwicklungsteams eingesetzt werden, sind die Resultate im Allgemeinen eine höhere Codequalität, weniger Regressionsbugs und stabilere Anwendungsbuilds. Tools wie die Codeanalyse- und Komponententestframeworks in Visual Studio 2008 stellen Entwicklungsteams eine automatisierte Unterstützung zur Verfügung, um sicherzustellen, dass der Prozess wiederholbar und vorhersagbar ist. Die Anwendung dieses Grads an automatisierter Strenge im Entwicklungsprozess führt mit jedem hinzugefügten Feature zur Nutzensteigerung. Wenn die Softwarekomplexität wächst, helfen automatisierte Testausführung und -abdeckung, das Tempo im Entwicklungsprozess zu bewahren und die Qualität der Anwendung aufrechtzuerhalten.

Prozesse verbessern und Standpunkte überprüfen

Bewährte Verfahren für Design, Programmierung, Debugging und Tests sind unschätzbare Werkzeuge für die Verbesserung der Softwarequalität. Das Zusammenfügen dieser verschiedenen Verfahren kann sich unter bestimmten Umständen für Entwicklungsteams als effektiv erweisen. In den meisten Fällen sind es jedoch die Prozesse und Standpunkte des Teams, die letztlich mit den bewährten Verfahren verschmelzen und eine Qualitätskultur ermöglichen, in der jedes Teammitglied Verantwortung für die Schaffung hochqualitativer Software trägt.

- **Entwicklungsprozesse verbessern** – Bugs werden über Tests sowie Debugging aufgespürt und beseitigt. Entwickler lernen aus diesen Fehlern und machen diese hoffentlich kein weiteres Mal, da die Bugs innerhalb von Bugverfolgungssystemen dauerhaft festgehalten und sehr detailliert dokumentiert werden. Dieser Prozess ist für Entwicklungsmethoden ebenso relevant. Erfolgreiche Teams überprüfen sich häufig selbst und nehmen Anpassungen für ihre zukünftigen Prozesse vor – in der Hoffnung, dass sie niemals die Fehler aus der Vergangenheit wiederholen werden. Die Implementierung von Änderungen,

welche die Effizienz im Entwicklungsprozess verbessern, führen letztendlich zur Qualitätssteigerung der geleisteten Arbeit. Verfahren wie die Einrichtung von Meilensteinen, Einchecken und Releasekriterien zur Qualitätssteigerung sind nur einige Beispiele, wie Prozesse mit Verfahren kombiniert werden können, um den Kreis zu schließen bei der Frage, wie sich solider Code ohne schwergewichtige Entwicklungsprozesse erzielen lässt.

- **Der Standpunkt ist alles** – Das Schreiben von fehlerfreiem Code ist mehr als nur das Einhalten einer vorgeschriebenen Liste bewährter Verfahren und Richtlinien. Es geht auch um die Dynamik von Entwicklungsteams, die Interaktionen mit den Kunden und Partnern sowie die Tatkraft, welche die Entwickler täglich aufbringen, um ihren wirklich sehr schwierigen Job zu verrichten. Es gibt bedeutende menschliche Faktoren, die uns jeden Tag bei der Entwicklung von Software beeinflussen, und es ist wichtig zu verstehen, wie man in diesen von hoher Dynamik geprägten Umgebungen gute Arbeit leistet.

Zusammenfassung

Verbesserungen in Programmiersprachen und RAD-Tools wie .NET und Visual Studio treiben die Industrie beständig an, qualitativ hochwertigere Software schneller, preiswerter und in Verbindung mit häufigeren Upgrades und Aktualisierungen zu liefern. Ungeachtet dieser fortgesetzten Nachfrage nach mehr Software und der Weiterentwicklung in unseren Tools und Prozessen bleiben das Erstellen und Veröffentlichen qualitativ hochwertiger Software in der Verantwortung des Entwicklers und stellen weiterhin eine Herausforderung dar. Wir als Softwareentwickler bemühen uns kontinuierlich um Verbesserungen an unseren Methoden und Verfahren, da wir bei der Erstellung hochqualitativer Software hohe Anforderungen von unseren Managern, Unternehmen und Kunden erfüllen müssen.

Es gibt viele ausgezeichnete Projektmanagementsysteme, Tools und Entwicklungsverfahren, die sich bei erfolgreicher Anwendung in Hinblick auf die Qualitätssteigerung und die Lieferung von Software als sehr effektiv erweisen. Aber die Kombination aus Qualität und Agilität lässt sich am besten erzielen, wenn ausgezeichnete Entwicklungsverfahren mit der Lieferstrenge und dem Lieferrhythmus verknüpft werden, die am besten mit den Erfordernissen Ihres Teams harmonieren. Während sich die Definition von Agilität in puncto Softwareerstellung je nach Unternehmen, Team und Softwaremarkt unterscheidet, gibt es eine Basisreihe aus Entwicklungsverfahren für das Schreiben von verwaltetem Code, die allgemein die Qualität der von Ihrem Unternehmen hergestellten Software verbessert. All diese Verfahren werden in nachfolgenden Kapiteln dieses Buchs ausführlich untersucht.

Das Wichtigste in Stichpunkten

- Es ist heutzutage eine Vielzahl an Softwareentwicklungsmethoden verbreitet. Eine ganze Reihe von ihnen, sowohl solche auf Wasserfall- als auch auf agiler Basis, stellen ausgezeichnete Gerüste für den Softwareentwicklungsprozess zur Verfügung und haben einen hohen Nutzen. Beide Arten haben das gleiche Ergebnis: Sie erlauben die schnelle Herstellung hochqualitativer Software für Ihre Kunden. Wählen Sie die Methode aus, die am besten zur Kultur und zu den Anforderungen Ihres Unternehmens passt.
- Agile Methoden setzen einen starken Schwerpunkt auf die qualitative Steigerung im Softwareentwicklungsprozess. Viele von diesen empfehlen Verfahren, die – einzeln oder in Kombination – früh im Entwicklungsprozess angewendet werden sollten.

- Qualitative Verfahren sollten früh (vorgezogen) im Entwicklungsprozess zur Anwendung gelangen, um zu erreichen, dass
  - Probleme mit dem Design oder Code in einem frühen Stadium entdeckt werden,
  - Entwickler ausreichend Zeit haben, entsprechend mit Code- und Designänderungen zu reagieren,
  - Tester eine hinreichend vollständige Testabdeckung sicherstellen können,
  - Zeitpläne generell und die termingerechte Abgabe des Produkts im Besonderen nicht gefährdet werden.
- Setzen Sie die folgenden Verfahren während der Entwicklung Ihrer Softwareanwendungen ein:
  - Integrieren Sie Klassendesign und Prototyping.
  - Setzen Sie Metaprogrammierung ein, um die Flexibilität und Wartungsfreundlichkeit zu erhöhen und die Komplexität zu verringern.
  - Legen Sie einen hohen Wert auf Performance, Skalierbarkeit und Sicherheit.
  - Lernen Sie, wie Sie den Arbeitsspeicher effektiv verwalten.
  - Nutzen Sie defensive Programmierung.
  - Verwenden Sie proaktive Debugroutinen.
  - Integrieren Sie automatisierte Komponententests und wiederholbare Testprozesse.
  - Implementieren Sie Testabdeckung.