Nico Orschel

(nico.orschel@aitgmbh.de)
ist MVP (Microsoft Most Valuable Professional), ISTQB-zertifizierter Software-Tester und Berater des AIT TeamSystemPro-Teams. Er hat sich u. a. auf das automatische Testen mit dem Visual Studio Lab Management spezialisiert. Er berät Unternehmen bei der Einführung und Anpassung des Visual Studio Team Foundation Server. Seine Erfahrungen vermittelt er zudem als Autor des TFS-Blogs, in Magazinen und in Vorträgen.

Links und rechts des Weges: Qualitätssicherung ist mehr als Testfallverwaltung

Testen ist Teil eines komplexen Prozesses auf dem Weg von einer Idee oder einem Problem zu einer fertigen Software. Testen hat die Aufgabe, die Qualität der und das Vertrauen in die auszuliefernde Software sicherzustellen. Dabei kommen verschiedene Testtechniken und -methoden zum Einsatz: funktionale Tests, Beta- und Alpha-Tests, Akzeptanztests sowie entwicklernahe Testtechniken, wie Unit- und Integrationstests sind nur einige Beispiele.

Als Technologie- und Prozessberater für effizientere Software-Entwicklung bei AIT haben wir häufig die Gelegenheit, die Software-Entwicklungsprozesse in verschiedenen Unternehmen zu analysieren. Ein Muster lässt sich dabei sowohl bei agil (z. B. Scrum) als auch bei traditionell (z. B. Wasserfallmodell) orientierten Arbeitsweisen beobachten. Viele Teams sind geprägt von einer strikten organisatorischen und thematischen Isolation von Testern und Entwicklern. Qualitätssicherung als Begriff ist in diesen Teams im Wesentlichen mit (manuellem) Testen gleichgesetzt (vgl. [Cha-1]). Der gelebte Prozess läuft meist nach einem ähnlichen Schema ab: zu Beginn Definition von Testfällen, dann (manuelle) Ausführung von Testfällen, Meldung von Fehlern an die Entwicklung, Nachtesten von Fehlerbehebungen und zum Abschluss Freigabe der Software.

Die Folgen dieser einseitigen Betrachtung von Qualitätssicherung als „nur Testen" sind immens. Es entstehen hohe Testaufwände bei geringer Qualität, verzögerte Freigabetermine aufgrund zu spät durchgeführter Qualitätssicherungsmaßnahmen und besonders schwerwiegend: isolierte und demotivierte Teammitglieder im gesamten Produktteam. Die Ursache: Qualitätssicherung wurde nicht als gemeinsame übergreifende Aufgabe wahrgenommen.

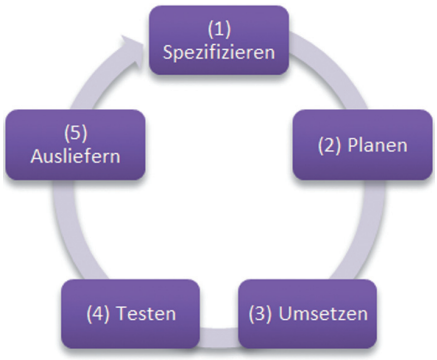

Abb. 1: Software-Lifecycle.

In den folgenden Abschnitten möchten wir Sie auf eine Reise durch die verschiedenen Bereiche eines Software-Lifecycles mitnehmen und Ihnen dabei neue Ideen links und rechts des Weges zur Verbesserung der Qualität Ihrer Software außerhalb der „klassischen" Testfallverwaltung und Ausführung von Testfällen aufzeigen. Im Detail werden dabei exemplarisch Maßnahmen für die Software-Lifecycle-Phasen „Spezifizieren", „Planen", „Umsetzen", „Testen" und „Ausliefern" betrachtet (siehe Abbildung 1).

Die Basis für alles: Anforderungen

Den Beginn einer jeden Software-Entwicklung markiert ein allgemein formuliertes Kundenproblem oder eine Anforderung. Diese unkonkreten „Kundenwünsche" werden anschließend durch einen formalen Anforderungsmanagement-Prozess strukturiert aufbereitet. Je nach Vorgehensmodell heißen die entstehenden Artefakte dann Anforderungen bzw. User Stories. Die Techniken zur Erfassung, Verarbeitung und Transformation dieser Kundenwünsche stellen hier nicht den inhaltlichen Fokus dar (weiterführende Informationen siehe [Sig-2]).

Aus unserer Sicht gibt es in vielen Projekten an der Schnittstelle zwischen Testen und Anforderungsmanagement die folgenden Ursachen für hohe Abstimmungsaufwände oder geringe Zusammenarbeit:

- **Medienbrüche:** Verwendung von getrennten Informationssystemen entlang des gesamten Entwicklungsprozesses: So verwendet z. B. das Produktmanagement Word-Dokumente, die Entwicklung SVN und die Tester HP QC für die Testverwaltung sowie Bugzilla für die Fehlererfassung und Kommunikation mit der Entwicklung. Das sorgt für Informationsverlust an jeder Schnittstelle.
- **Unklare Verhältnisse:** Reviews werden selten durch wichtige Interessensvertreter (Stakeholder) durchgeführt. Die

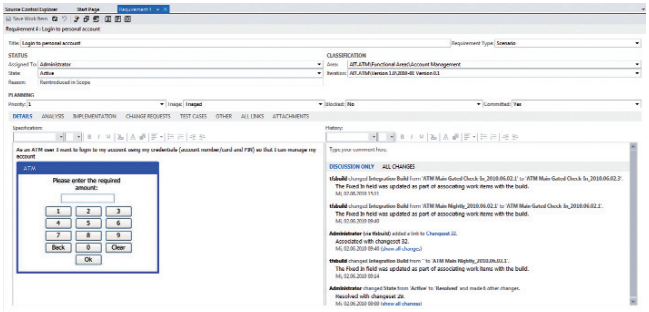

Abb. 2: Anforderungen aufnehmen mit Visual Studio.

Interessensvertreter sind gegensätzlicher Meinung und Ansicht und es ist nicht geklärt, wer in letzter Instanz entscheidet. Das sorgt für fehlende Klarheit und häufiges Verfehlen der Erwartungen.

- **Fehlende Informationen**: Entscheidungen und Review-Ergebnisse sind nicht dokumentiert oder stehen aufgrund unterschiedlicher Werkzeuge nicht zentralisiert für alle zur Verfügung.

Eine Möglichkeit, diese Probleme anzugehen, ist die Nutzung einer integrierten ALM-Plattform. Solche Plattformen decken dabei verschiedene ALM-Bereiche wie Software-Lifecycle-Management, Betrieb (Operations) und Verwaltung (Governance) ab (Definition von ALM siehe [Cha-2]). Für die Entwicklungsprojekte mit unseren Kunden setzen wir dabei seit mehreren Jahren erfolgreich auf den Team Foundation Server (TFS) aus dem Hause Microsoft. Für die Phase „Anforderungsmanagement" ermöglicht der TFS über so genannte „Work Items" die einfache Erfassung und Verwaltung von Anforderungen oder User Stories. Über den gleichen Mechanismus werden auch Testfälle, Feedback und Reviews (inkl. Ergebnisse) in den späteren Phasen unterstützt. In Abhängigkeit vom jeweiligen Artefakttyp („Anforderung", „Review", „Feedback") können die Felder und Regeln sowie der Workflow variieren. Die jeweiligen Elemente sind keine geschlossenen Systeme, sondern können im laufenden Projekt an die jeweiligen Bedürfnisse angepasst werden. In **Abbildung 2** ist beispielhaft die Erfassung einer Anforderung in Visual Studio dargestellt. Neben Visual Studio können Sie als Anwender auch andere Werkzeuge gleichberechtigt verwenden. Beispiele sind z. B. Webzugriff per WebAccess, Excel, Project, Visual Studio, Microsoft Testmanager (MTM), AIT WordToTFS (siehe [Ait]) oder TeamCompanion für Outlook. Der große Vorteil besteht darin, dass je nach Rolle der Teammitglieder bereits etablierte Werkzeuge für den Zugriff auf die gemeinsamen Artefakte wie Anforderungen, Fehler oder Feedback genutzt werden können. Durch die zentrale Speicherung aller Informationen im TFS haben alle Teammitglieder die Möglichkeit, zu jeder Zeit Rückmeldungen/Verbesserungsvorschläge zu allen Artefakten (hier: Anforderungen) zu geben. Die erhöhte Transparenz sorgt als positiver Effekt dafür, dass Inkonsistenzen, Lücken und Widersprüche bereits frühzeitig erkannt und behoben werden.

Ein Beispiel für einen gelebten Prozess aus unserem Projektalltag ist die Nutzung von Word und TFS für die Anforderungsdefinition. In der Spezifikationsphase setzen wir primär auf Word mit WordToTFS, unsere kostenlosen Erweiterung für Word und TFS. WordToTFS ermöglicht die einfache und schnelle Aufnahme, Bearbeitung und Synchronisierung von Work Items zwischen Word und dem TFS (siehe **Abbildung 3**). Die Synchronisierung kann dabei sowohl von Word nach TFS als auch von TFS nach Word stattfinden. Der Einsatz von Word hat dabei mehrere praktische Vorteile: Erstens kann das Anforderungsdokument sehr einfach als Vertragsbestandteil verwendet werden, zweitens können Kunden auch ohne TFS-Zugriff problemlos Anforderungen prüfen und anpassen und drittens steht allen Beteiligten der aktuelle Stand im TFS zur Verfügung.

Im nächsten Schritt („Planen") werden aus den hinterlegten Anforderungen die jeweiligen Aufgaben für alle Teammitglieder erstellt, geplant und mit den betroffenen Anforderungen verknüpft. Hier führt ebenfalls wieder Transparenz zur frühzeitigen Vermeidung von Fehlern und Missverständnissen.

Mit Ausblick auf den neuen TFS 2012 sollen zwei neue Programme für die frühen Entwicklungsphasen nicht unerwähnt bleiben: Feedback-Client und PowerPoint-Storyboarding.

Der Feedback-Client ermöglicht die Aufzeichnung von Feedback durch Stake-

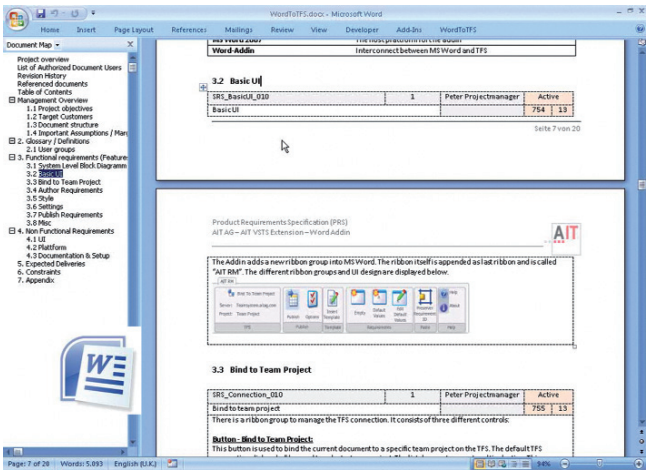

Abb. 3: Anforderungen aufnehmen mit WordToTFS.

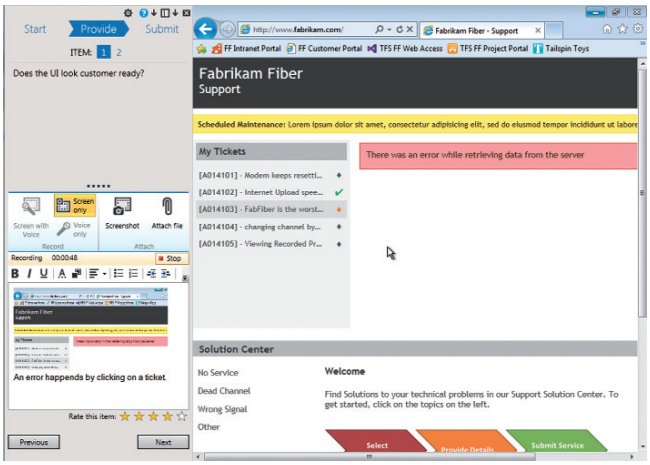

Abb. 4: Feedback an Entwicklung geben.

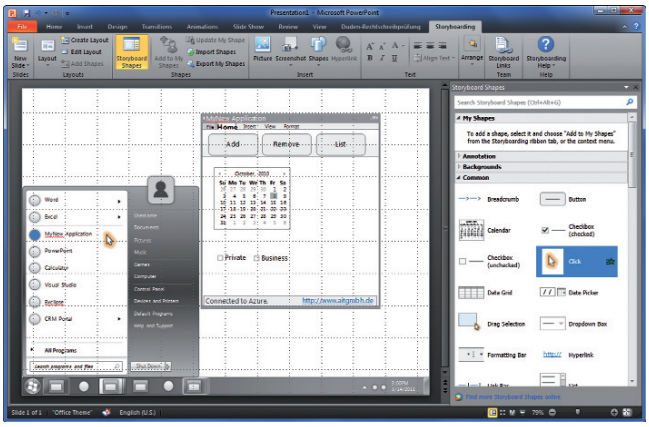

Abb. 5: Storyboarding mit PowerPoint.

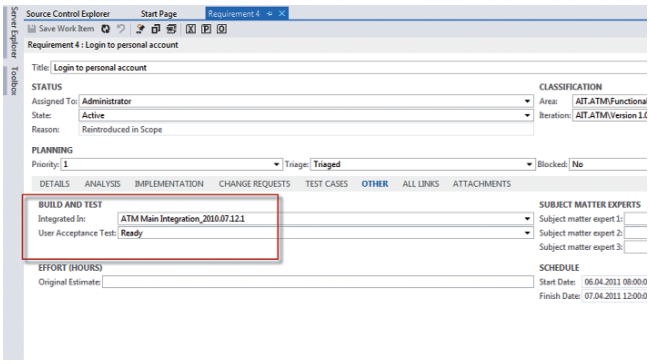

Abb. 7: Nachverfolgung von Anforderungen zu Builds.

holder. Das Programm leitet den Anwender durch einen einfachen Prozess, bei dem Screenshots oder eine Videoaufzeichnung erzeugt werden, während der Anwender die Testversion einer Software oder Webseite prüft – ein Bild sagt schließlich mehr als 1000 Worte (siehe Abbildung 4). Das Feedback wird direkt im TFS gespeichert und kann Grundlage für die Änderung von Anforderungen sein.

Das PowerPoint-Storyboarding als weiteres Instrument ermöglicht die Erstellung von UI-Prototypen mit PowerPoint (siehe Abbildung 5). Durch Verwendung von PowerPoint als Standardwerkzeug können mit sehr wenig Lernaufwand und ohne Code frühzeitig Prototypen eines Programms erstellt werden. Auf Grundlage dieser Prototypen können Rückmeldungen von wichtigen Entscheidungsträgern oder Kunden eingeholt werden, noch bevor die Entwicklung in die falsche Richtung startet. Die erstellten Storyboards wiederum werden zentral abgelegt und mit den Anforderungen im TFS verknüpft. Sie stehen damit für die spätere Verwendung neben den Anforderungen zum besseren Verständnis zur Verfügung.

Nachverfolgung von Anforderungen

Nachdem die Anforderungen an das Produkt definiert und Aufgaben geplant sind,

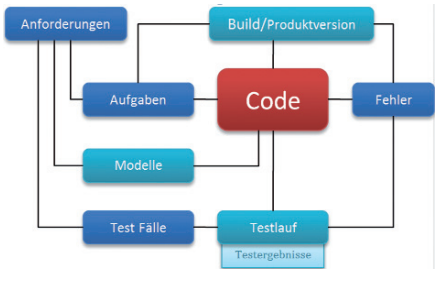

Abb. 6: TFS-Beziehungsmodell.

steht als großer Schritt deren Umsetzung an. Auch hier gibt es oft Stellen, an denen es zwischen Testern und Entwicklern häufig zu Problemen kommt:

- **Fehlende Nachverfolgbarkeit** von Quellcode zu anderen Projektartefakten (z. B. Anforderungen/Aufgaben).
- **Stand der Umsetzung** ist für Nicht-Entwickler nicht transparent.
- **Übergabe** von halbfertigen Funktionen an andere Interessengruppen (z. B. Tester).
- **Integration** von Teillieferungen innerhalb der Entwicklung erfolgt zu spät für Tests.

Auch für den Bereich Quellcodeverwaltung setzen wir bei AIT den TFS als Plattform ein. Der TFS bietet die Möglichkeit, an einen Eincheck-Vorgang bestimmte Bedingungen zu stellen (Check-In Policies), welche vor jedem Check-In von Änderungen erfüllt sein müssen. Über diese Regeln kann man z. B. den Entwickler erinnern, bei jedem Eincheck-Vorgang die bearbeitete Anforderung zu verknüpfen, um eine Nachverfolgung zwischen Code und Anforderung (siehe Abbildung 6) zu ermöglichen.

Der TFS bietet die Möglichkeit, diverse Artefakte miteinander zu verknüpfen, sodass ein umfangreiches Beziehungsgeflecht der verschiedenen Artefakte wie in Abbildung 5 entsteht. Auf Basis dieser Informationen hat der Tester jetzt Möglichkeiten, den Stand einer Umsetzung bzw. Fehlerbehebung einzusehen oder die Verknüpfung zur Durchführung von Auswirkungsanalysen z. B. zwischen Code und Anforderungen oder Code und Testfällen zu verwenden. Ein häufiger Anwendungsfall ist die Ermittlung, ob ein Feature bereits fertig gestellt wurde und in welches Release es integriert wurde. In Abbildung 7 sehen Sie ein Beispiel für genau diese Nachverfolgung von einer Anforderung zu einem speziellen Build. Durch den Status der Anforderung ist der Stand der Umsetzung ebenfalls ableitbar. Zusätzlich kann der Entwickler nach Fertigstellung eines Features die Aufgabe ohne Medienbruch an den Tester weiterreichen.

Strukturierung der Quellcodeverwaltung

Im Themenbereich der Quellcodeverwaltung gibt es noch einen weiteren großen Bereich, welcher die Tester direkt betrifft: die Struktur der Entwicklungszweige (Branches). Jetzt stellt sich sofort die Frage: Wo besteht der Zusammenhang zwischen Branching und Testen? In vielen bestehenden Projekten findet man sehr oft eine Struktur wie in Abbildung 9 dargestellt. Diese Struktur ist gekennzeichnet durch eine Hauptentwicklungslinie („main“) sowie einen oder mehrere „release“-Zweige für die veröffentlichten Versionen. Jeder dieser Entwicklungszweige ist ein in sich konsistenter Container mit allen zusammengehörigen Produktartefakten (Quellcode, Tests, Bibliotheken, Build-Skripts, Dokumentation etc.). Auf der genannten Hauptentwicklungslinie arbeiten dabei sowohl Entwickler als auch Tester. Entwickler setzen fortlaufend neue Features um und checken diese direkt in den Hauptzweig ein. Die Tester wiederum bedienen sich zum Testen jeweils einer Software-Version aus diesem Zweig. Man kann daraus schlussfolgern, dass die Quellcodeverwaltung eine direkte Schnittstelle zwischen Entwicklung und Tester und entsprechend kritisch für die Arbeit von beiden Parteien ist. Das aktuelle Vorgehen mit „main“- und „release“-Zweigen hat auf den ersten Blick den Charme, dass es sehr leichtgewichtig scheint und Mehr-

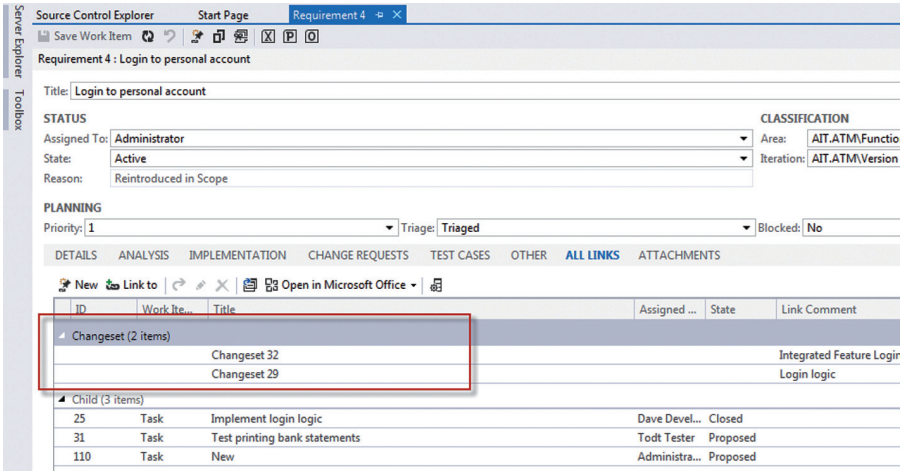

Abb. 8: Verknüpfung von Anforderungen und Changesets.

arbeit verhindert. Auf den zweiten Blick aber offenbaren sich in vielen alltäglichen Situationen jedoch Schwächen. In einigen Projekten beobachtet man die Situation, dass an Tester öfter unfertige Features übergeben werden und anschließend eine Welle von „unnötigen" Fehlermeldungen an die Entwicklung gemeldet wird. Diese „unnötigen" Fehlermeldungen sorgen als Folge dann für unnötige Mehrarbeit und damit Demotivation auf beiden Seiten.

Das beschriebene Problem lässt sich dabei mit einem überschaubaren Einsatz durch eine angepasste Branching-Struktur adressieren. In der Praxis gibt es je nach Situation verschiedene Strategien (vgl. [Vsa]). In Abbildung 10 wurde dazu als Erweiterung ein zusätzlicher Entwicklungszweig („dev"-Branch) eingeführt. Die Weiterentwicklung neuer Features wird nun immer zuerst auf diesem Zweig erfolgen. Ist ein Feature vollständig implementiert und es erfüllt zuvor festgelegte Qualitätsanforderungen (Quality Gates), dann werden die Änderungen vom „dev"-Zweig nach „main" integriert. Beispiele für zuvor genannte Quality Gates sind:

- **Kompilieren** des Quellcodestands ist möglich.
- **Statische Code-Analyse** wurde ausgeführt und Regelverletzungen befinden sich im tolerierten Bereich, d. h. der Quellcode wurde automatisch auf Code-Konventionen geprüft und entspricht damit den internen (Projekt-) Vorgaben.
- **Unit- und Integrationstests** sind erfolgreich ausgeführt worden.

Als Ergebnis der durchgeführten Anpassung enthält der „main"-Zweig nur noch stabile Versionen und damit testbare Features. Als Tester verwendet man jetzt nicht mehr „halbfertige" Versionen aus dem „dev"-Zweig, sondern nur stabile aus dem „main"-Zweig, sodass „unnötige" Fehlermeldungen und Blockaden seitens der Entwickler vermieden werden.

Automatisches Erstellen der Software

Zu guter Letzt verbleibt noch ein Punkt mit viel Diskussionsstoff: automatisiertes Kompilieren und Prüfen der Software mittels eines Build-Prozesses. Werden Versionsstände der Software nicht automatisiert, sondern manuell und auf einem „bestimmten Entwickler-PC" ausgeführt, dann ergeben sich folgende Risiken:

- Kompilieren ist nur mit komplexem Setup auf einem Entwickler-Rechner möglich und damit fehleranfällig und aufwändig.
- Wissen über die Release-Erzeugung steckt in wenigen Köpfen und ist in aller Regel nicht dokumentiert.
- Prüfung, ob abhängige Produkte noch funktionieren, erfolgt sehr spät (Integrationsfehler).
- Integration von nicht autorisierten Software-Versionen verursacht Fehler bei Kunden und beim Testen.
- Statische und dynamische Tests werden aus Zeitgründen nicht ausgeführt (Unit-Tests, statische Code-Analysen etc.).

Auch zur Abbildung von Build-Prozessen setzen wir auf die in den TFS integrierte Build-Plattform. Der TFS unterscheidet zwischen fünf Build-Typen: „Continuous Integration", „Rolling Builds", „Gated Check-In Builds", „Scheduled Builds" und „Manuell gestartete Builds". Die Kunst besteht jetzt darin, diese verschiedenen Build-Typen in Abhängigkeit von der jeweiligen Entwicklungslinie und Zielstellung einzurichten. Nicht jeder Build-Typ ist für jede Situation geeignet. Die richtige Mischung zwischen Branch- und Build-Typ hingegen kann statt Mehrarbeit die Qualität ihrer Releases signifikant steigern. Mehrarbeit wird reduziert, weil auch hier sehr früh automatisch Feedback an die Entwickler gegeben wird. Neben der Funktion als Feedback-Instrument bilden Build-Prozesse zudem eine unabhängige Prüfinstanz zwischen den verschiedenen Entwicklungszweigen bzw. -phasen.

Eine Empfehlung aus unserem Projektalltag ist der Einsatz von Continuous Integration Builds (CIBs) auf der Entwicklungslinie für schnelles Feedback. Dabei löst jeder Check-In automatisch den Build der Software aus. Der Vorteil besteht darin, dass jede Änderung geprüft wird und Fehler auf Ebene des einzelnen Check-Ins

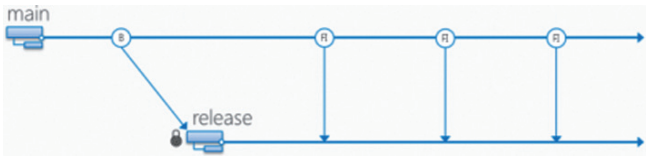

Abb. 9: Einfache Branching-Struktur.

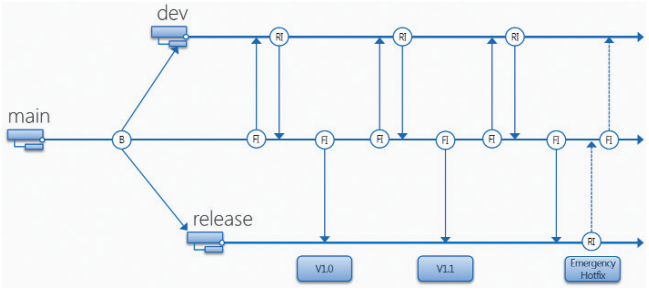

Abb. 10: Basis-Branching-Struktur.

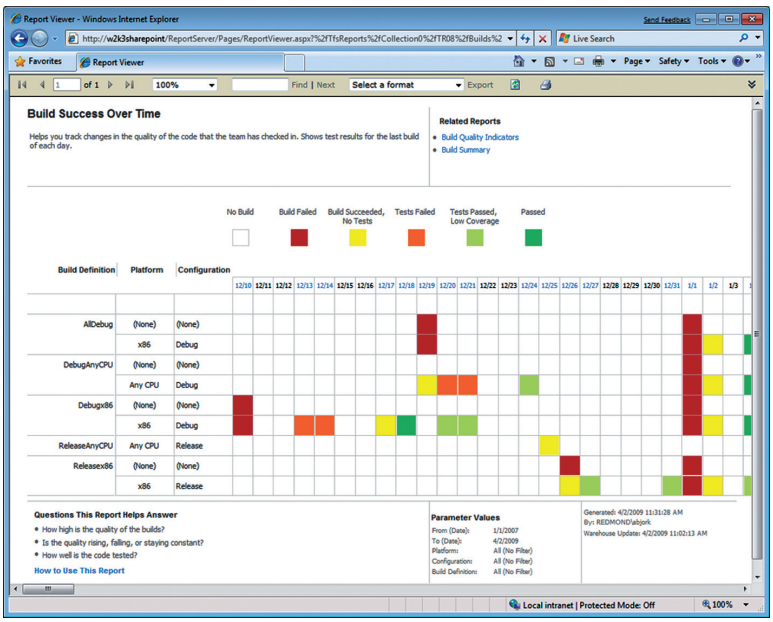

Abb. 11: Build Sucess Over Time.

schnell identifiziert werden können. Der Rolling Build als Besonderheit fasst im Wesentlichen mehrere Eincheck-Vorgänge in einem Build-Durchlauf zusammen, sodass sich Build-Zeiten verringern lassen.

Als nächster Schritt wird die Hauptlinie („main“) mit einem täglich laufenden Build-Prozess (Scheduled Build) weiteren automatischen Prüfungen, wie statischen Code-Analysen (z. B. mit FxCop oder StyleCop), weiteren automatisierten Tests (z. B. Integrationstests), Sandcastle-Dokumentationserstellung oder Setup-Erstellung, unterzogen. Der tägliche Build fungiert damit für das Team als regelmäßiger umfassender „Gesundheitscheck“. Durch die tägliche Ausführung eignen sich die erfassten Kennzahlen (Code Coverage, statische Code-Analyse, Anzahl geänderten Codes, etc.) auch gut, um über Reports den Ist-Zustand sowie den Trend bzgl. der Qualität beurteilen zu können. Ein Beispiel für einen Build-Test-Report zeigt Abbildung 11. Neben der reinen Gesundheitsprüfung ist dieser spezielle Build auch für die Erstellung der Testversion der Software zuständig. Aufgrund der vielfältigen Vorprüfungen hat diese Version fast schon einen „Release“-Charakter.

Eine TFS-Besonderheit bei den Builds bildet der Gated Check-In Build. Hier wird im Gegensatz zum CI-Build ein Build vor dem jeweiligen Check-In-Vorgang erzwungen, und im Fehlerfall werden Änderungen der Entwickler vom Server abgelehnt. Ein CI-Build wird hingegen immer erst ausgelöst, wenn Änderungen in der Versionskontrolle bereits eingecheckt sind. Dadurch, dass Gated Check-In Builds nicht umgangen werden können und Änderungen vor dem Check-In geprüft werden, können kritische Bereiche wie z. B. „release“-Zweige, auf denen nur ab und an eine Fehlerbehebung erfolgt, besonders geschützt werden.

Integriertes Testmanagement

Nachdem die Software erfolgreich erstellt wurde, kann diese an die Tester übergeben werden. In dieser Phase bietet es sich an, den Microsoft Test Manager als integriertes Werkzeug für Testmanagement- und Testausführungsaufgaben mit dem TFS einzusetzen. Vorteilhaft sind hier die tiefe Integration mit dem TFS und dadurch Vermeidung von Medienbrüchen mit den Systemen der Entwicklung. Im Vorgängerartikel (siehe [Sig-1]) wird detailliert auf diesen Testprozess mit MTM und die Optimierung der Kommunikation mit Entwicklern eingegangen.

Fazit

Wir haben Ihnen im Artikel gleich mehrere Ideen zur ganzheitlichen Steigerung der Qualität Ihrer Entwicklung aufgezeigt. Diese Ideen sind dabei bewusst nicht auf Testfallverwaltung und -ausführung beschränkt. Der Fokus der Verbesserungsmaßnahmen liegt schwerpunktmäßig bei Aktivitäten an den Schnittstellen Tester/Entwickler, Tester/Produktmanagement und Tester/Kunde. Als konkrete Maßnahme wurde gezeigt, dass ein integriertes System dabei helfen kann, frühzeitig Fehler bereits in der Anforderungsdefinitionsphase zu finden. Dies ist möglich, weil alle Anwendergruppen einen „einheitlichen“ Zugriff auf alle Informationen der Entwicklung haben – die „einzige Wahrheit“ des Projekts. Beispiele sind z. B. Anforderungen, Aufgaben, Testfälle oder Prototypen.

Zusammenfassend kann gesagt werden, dass Qualitätssicherung kein reiner Job der Tester ist, sondern alle Teammitglieder durch spezifische Maßnahmen zur Steigerung der Qualität beitragen können.

Sollten Sie Fragen und Anregungen zum dargestellten Prozess und den Werkzeugen haben, helfen wir Ihnen gerne weiter! ■

Referenzen

[Ait] AIT WordToTFS, siehe http://www.aitgmbh.de/downloads/kostenlose-tools/ait-wordtotfs-word2tfs.html.

[Cha-1] REDEFINING QUALITY ASSURANCE – AN ALM PERSPECTIVE, siehe http://www.davidchappell.com/writing/white_papers/Redefining_QA--Chappell.pdf.

[Cha-2] WHAT IS APPLICATION LIFECYCLE MANAGEMENT?, siehe http://www.davidchappell.com/writing/white_papers/What_is_ALM_v2.0--Chappell.pdf.

[Sig-1] Ausweg aus der Kommunikationskrise oder das Ende von „Bei mir funktioniert's“?, siehe http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2010/Testing/orschel_OS_TESTING_2010.pdf.

[Sig-2] OBJEKTspektrum Online-Themenspecials: Requirements Engineering, siehe http://www.sigs-datacom.de/fachzeitschriften/objektspektrum/online-themenspecials/artikelansicht.html?show=2920.

[Vsa] Visual Studio Team Foundation Server Branching and Merging Guide, siehe http://vsarbranchingguide.codeplex.com/releases.