# Agile Entwicklung mobiler Geschäftsapplikationen

Agile Softwareentwicklung verspricht bedarfsgerechtere Lösungen in kürzerer Zeit. Doch funktioniert dies wirklich, auch für mobile Applikationen? Lässt sich dieser Ansatz mit der Verwendung aktueller Softwaretechnologien verbinden? Der vorliegende Erfahrungsbericht vermittelt einen Eindruck davon. Peter K. Brandt

Seit einigen Jahren gilt agile Softwareentwicklung als schlankere und bedarfsorientiertere Methode zur Erstellung von Software im Vergleich zu schwergewichtigen Methoden wie beispielsweise der traditionellen Wasserfallmethodik. Hohe Effizienz durch Vermeidung unproduktiver bürokratischer Abläufe, die Möglichkeit, rasch auf Veränderungen eingehen zu können, und eine permanente Fortschrittskontrolle durch häufige Lieferungen des stets lauffähigen Arbeitsresultats sind nur einige Vorteile, die oft mit agiler Softwareentwicklung verbunden werden und höhere Erfolgswahrscheinlichkeiten mit sich bringen (siehe auch Agile Manifesto, http://agilemanifesto.org).

Ein grosses Schweizer Unternehmen beauftragte Ergon Informatik vor einigen Monaten, eine mobile Geschäftsanwendung für die Aussendienstmitarbeiter zu erstellen. Auftraggeber und Umsetzungsteam waren sich einig, dass für das Projekt ein agiler Ansatz zum Zuge kommen sollte. Insbesondere wurde gemeinsam geplant, das Projekt in mehreren kurzen Zyklen, sogenannten «Sprints», von jeweils vier Wochen umzusetzen. Am Ende jedes Sprints lieferte das Entwicklerteam eine lauffähige Teilversion der zu erstellenden Applikation, die im Lauf der Projektzeit immer vollständiger wurde. Jeweils zu Sprintbeginn wurde mit den Auftraggebern vereinbart, welche Requirements innerhalb eines Sprints umgesetzt werden sollten. Auf diese Weise konnten die Auftraggeber diejenigen Features zuerst realisieren lassen, die höchste Priorität hatten, und diese auch bereits frühzeitig gründlich auf ihre Praxistauglichkeit validieren. Gewonnene Erkenntnisse konnten dabei kurzfristig in die Planung des nächsten Sprints einfliessen.

Das Umsetzungsteam auf der anderen Seite erhielt auf diese Art sehr früh Feedback über die Erwartungserfüllung und Korrektheit der Applikation. Der daraus resultierende Mehrwert war so gross, dass bald der Entscheid fiel, die Dauer der einzelnen Entwicklungssprints sogar auf zwei Wochen zu verkürzen, um die Agilität weiter zu erhöhen – sprich noch zeitnaher Feedback zu bekommen und einfliessen zu lassen.

**Peter K. Brandt** ist Lead Engineer Mobile and Embedded bei der Ergon Informatik AG in Zürich.

Die mobile Geschäftsapplikation musste sich in die IT-Umgebung des Auftraggebers einfügen und sollte für die Android-Plattform erstellt werden. Sie verwendet Server- und Hintergrundsysteme, die von einer Gruppe weiterer Anbieter geliefert werden. Die dazu benötigten Schnittstellen waren zu Projektbeginn noch undefiniert und ein erwartetes Arbeitsresultat.

Bei der Softwareentwicklung wurden die nachstehenden Ansätze verfolgt, um in der beschriebenen Ausgangslage auf agile, iterative Weise zu zuverlässigen Resultaten zu gelangen, die den Anforderungen der Auftraggeber gerecht werden:

- Continous Integration und automatisierte Erstellung von Releases: Erleichtert die regelmässige Lieferung aktueller Softwareversionen an die Auftraggeber ohne manuelle Aktivitäten.
- Konsequente Code-Reviews: Hohe Softwarequalität durch Einhalten des Vier-Augen-Prinzips.
- Testgetriebene Entwicklung und automatisierte Testausführung: Dies stellt sicher, dass der Code stets den aktuellen Anforderungen entspricht, selbst wenn sich diese ändern, und dass trotz neuer Features bestehende voll funktionsfähig bleiben.
- Klare und exakt beschriebene Schnittstellendefinitionen und verbindliche, ausführbare Tests für deren Einhaltung: Reduktion der Abhängigkeiten zwischen den Systemkomponenten in dem sich rasch wandelnden Umfeld.
- Mock Server: Entkopplung der Softwareentwicklung zwischen den verschiedenen, räumlich getrennten Entwicklungsteams für die einzelnen Systemkomponenten.

## Continous Integration (CI)

Während der Erstellung der Software legt der Entwickler in sich abgeschlossene Mengen von Artefakten (Programmcode, Konfigurationsdateien, Icons etc.) versioniert im Code-Repository ab. Der Continous-Integration-Server überwacht dieses Repository. Im vorliegenden Projekt kam hierzu der CI-Server «Jenkins» zum Einsatz. Nach jeder Veränderung holt Jenkins die aktuelle Version der Artefakte in ein leeres Verzeichnis und führt dort einen kompletten Build der Software aus. Auf diese Weise wird sichergestellt, dass jede im Repository vorhandene Version korrekt übersetzt werden kann und nicht beispielsweise von Dateien oder Einstellungen auf dem Rechner des Entwicklers abhängig ist. Anschliessend werden auf der aktuellen Softwareversion alle automatisierten Tests ausgeführt (siehe unten). Mögliche Probleme werden dem Team unmittelbar mitgeteilt, sodass sofort reagiert werden kann. Falls alle zu bestehenden Testfälle erfolgreich absolviert wurden, wird ebenfalls vollautomatisch eine Archivdatei mit dem Softwarerelease erstellt, die dem Auftraggeber ausgehändigt werden kann. Das Erzwingen erfolgreicher Tests stellt sicher, dass keine Releases geliefert werden, die den Anforderungen nicht genügen.

## Konsequente Code-Reviews

Nachdem ein Entwickler ein bestimmtes Feature erstellt hat, beauftragt er ein anderes Teammitglied mit dem Review aller neuen oder geänderten Artefakte. Dazu kommt einerseits das oben genannte Code Repository zum Einsatz, andererseits auch ein spezielles Tool für Code-Reviews, mit dem der Reviewer an den Artefakten auf einfache Weise Kommentare oder Verbesserungsanregungen anbringen kann, die der Originalautor erhält und berücksichtigt. Durch dieses Vier-Augen-Prinzip wird die Codequalität weiter erhöht und Fehler und Ungenauigkeiten, die von den Testfällen nicht abgefangen wurden, können aufgedeckt werden.

## Testing, Testing, Testing

In diesem Projekt wurden automatisierte Testfälle auf drei Ebenen eingesetzt:

**1. Modultests (Unittests)** stellen sicher, dass die einzelnen Komponenten der Software den vom Entwickler beabsichtigten Zweck erfüllen. Zur Erstellung der Modultests für die mobile Geschäftsapplikation für das Android-Betriebssystem kamen die Testframeworks JUnit, Robo-

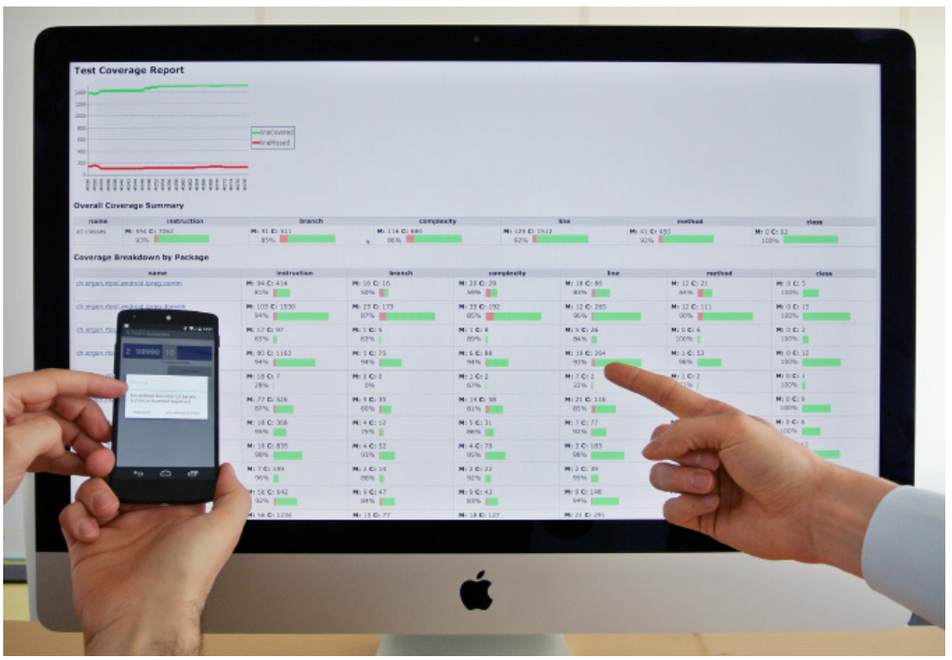
lectric und Mockito zum Einsatz. Bei der Implementierung wurde darauf geachtet, dass nach Möglichkeit zunächst die Testfälle erstellt wurden, bevor die eigentliche Softwarekomponente programmiert wurde. Dieses Vorgehen, auch «Test Driven Development» genannt, hat zur Folge, dass eher Softwarekomponenten entstehen, die nur lose gekoppelt sind, da sie so einfacher testbar sind. Das «Design for Testability» hat also neben der einfacheren Erstellung von Testfällen auch einen positiven Nutzen auf die exakte Trennung der Softwarekomponenten in klare Zuständigkeiten. Dadurch sind jene wiederum leichter verständlich, weniger fehleranfällig und einfacher wart- und erweiterbar.

**2. Tests am Benutzerinterface:** Die UI-Tests («User Interface Tests») machen das, was eine Testperson manuell an der mobilen Geschäftsapplikation ausführen würde, um diese zu testen. Im Projekt kam zur Automatisierung dieser Tests das für die Android-Plattform entwickelte Testing-Framework «Espresso» zum Einsatz. Damit kann eine mobile App vollautomatisch über ihr echtes Benutzerinterface getestet werden. So kann mit wenig Aufwand bei jedem Release ein automatischer Kompletttest der Anwendung inklusive vieler Grenzfälle erfolgen, der mit manuellen Tests weniger zuverlässig und deutlich kostenintensiver wäre und daher oft ausbleiben würde.

Modultests und UI-Tests bilden zusammen die Basis für eine zuverlässige Qualitätssicherung des Programmcodes. So wird erreicht, dass zukünftige Anpassungen einzelner Entwickler nicht an anderen Stellen der Applikation unerwünschte Konsequenzen oder gar Fehler zur Folge haben.

Wir hatten zudem bereits von Beginn des Projekts an darauf geachtet, konstant eine sehr hohe Testabdeckung zu erreichen. Dadurch entwickelte sich das Softwaredesign auf natürliche Weise in eine Richtung, die das Erstellen weiterer Tests für neue Komponenten einfach möglich macht. Würde man hingegen erst später im Projekt versuchen, nachträglich aussagekräftige Testfälle zu erstellen, so wird dies oft durch unpassende Voraussetzungen und ein unglücklich gewähltes Softwaredesign erschwert.

**3. Schnittstellentests:** Die Schnittstellen zu den im Projektumfeld vorhandenen Umsystemen wurden einerseits in einer versionierten Schnittstellenspezifikation in menschenlesbarer Form festgehalten. Da die Erfahrung zeigt, dass auch textuelle Beschreibungen von Schnittstellen gerade in Grenzfällen häufig Interpretationsspielraum über das erwartete Verhalten haben, wurden zusätzlich automatisierte Schnittstellentests erstellt. Diese Testfälle greifen auf eine Schnittstelle zu, rufen sie mit vorgegebenen Parametern auf und prüfen, ob das zurückgelieferte Ergebnis der Erwartung des Testautors entspricht. Da die Schnittstellentests auf diese Art in einer formalisierten und einfach ausführbaren Form vorliegen, kann die Einhaltung des Schnittstellenvertrags durch beide beteiligten Seiten einfach überprüft werden.



Analysewerkzeuge ermitteln den Grad der Testabdeckung. Bild: Ergon

Da die Serversysteme der Applikation zunächst noch nicht verfügbar waren, wurde zudem vom Entwicklungsteam ein «Mock-Server» erstellt, der das Verhalten der realen Schnittstelle simuliert. Dieser Mock-Server ahmt das Verhalten des echten Servers nach, hat aber keine realen Auswirkungen auf weitere Systeme. Mithilfe der Schnittstellentests konnte sichergestellt werden, dass sich auch der Mock-Server so verhält, wie sich später der echte produktive Server verhielt. So konnte mit der Entwicklung der Applikation sofort begonnen werden. Diese nutzte zunächst den Mock-Server, wodurch sie auch vom Auftraggeber bereits komplett getestet werden konnte. Dank der ausführbaren Schnittstellentests brachte die spätere Umstellung auf den echten Server keine Überraschungen mehr: Die Applikation verhielt sich wie erwartet.

## Folgenutzen: End-to-End-Überwachung des produktiven Betriebs

Inzwischen geht die Applikation in den produktiven Betrieb über. Dabei haben sich die oben genannten UI-Tests mit dem Espresso-Framework erneut als nutzbringend erwiesen: An einer speziellen Teststation kann ein Mobiltelefon angeschlossen werden, auf dem in kurzen, regelmässigen Intervallen aussagekräftige UI-Tests ausgeführt werden, gerade so, als ob ein Mensch das Gerät manuell bedienen würde. Falls das Espresso-Framework die erwartete Reaktion nicht feststellen kann, löst es einen Alarm über das vorhandene Monitoringtool aus, sodass die Fehleranalyse und -behebung durch den Betrieb bereits erfolgen können, bevor die Benutzer vom Problem betroffen sind.

Da die Tests auf dem echten Endgerät stattfinden und über das reale Netz auf die produktiven Backend-Server zugreifen, ist auf diese Weise eine komplette End-to-End-Überwachung sichergestellt.

## Fazit: Hohe Qualität und Zuverlässigkeit

Die mobile Android-Geschäftsanwendung ist ein wichtiges Arbeitswerkzeug für mehrere tausend Mitarbeiter. Dank der konsequenten Anwendung des agilen Vorgehens und der beschriebenen Entwicklungsmethoden wurde eine hohe Qualität und Zuverlässigkeit erreicht. Die starke Automatisierung von Testausführung und Release-Erstellung im Rahmen von Continous Integration garantiert reproduzierbare Resultate.

Beim Auftraggeber führten das Miterleben der stetig wachsenden Applikation und die immer vorhandene Möglichkeit der Steuerung zu einer grossen Zufriedenheit mit dem Endresultat. Es hat sich erneut gezeigt, dass auch bei der Entwicklung mobiler Anwendungen agile Methoden zu einer hohen Qualität des Ergebnisses und erfreulicher Kundenzufriedenheit führen.