CI - Dauerhaft integriert entwickelt es sich schneller

Sören Halter
Oracle B.V. & Co. KG
Dreieich

Schlüsselworte
Softwareentwicklung, Wasserfallmodell, Continuous Integration, Kontinuierliche Integration, Permanente Integration, Extreme Programming, Agile Development, Subversion, Maven, Hudson, Automatische Tests, Oracle JDeveloper, Oracle SOA Suite, Oracle Java Cloud Service, Oracle Developer Cloud Service.

Einleitung
Die kontinuierliche Integration (engl. Continuous Integration; CI) ist seit Extreme Programming (XP) der neue heilige Gral der Softwareentwicklung, da er durch eine fortlaufende Integration der Komponenten aller Entwickler die durch Phasenmodelle erzeugte Integrationshölle vermeidet. Mit Tools wie Hudson, Jenkins, Maven, CVS, Subversion, Git, JUnit etc. lassen sich Lösungen umsetzen, die eine kontinuierliche Integration vollständig automatisieren.
Der weitere Vortrag zeigt auf, warum kontinuierliche Integration sinnvoll ist, wie eine Umgebung hierfür im Kontext der Entwicklung in Oracle Middleware aussehen kann und wie durch diese die Identifikation typischer Regressionsfehler erleichtert wird.

Probleme mit traditionellen Entwicklungsmethoden

In klassischen Softwareentwicklungsmethoden, wie z.B. dem „Wasserfallmodell“, werden die einzelnen Komponenten, aus denen ein Softwaresystem besteht, nur selten zum Gesamtsystem zusammengebaut und im Zusammenspiel getestet. Stattdessen testet jeder Entwickler seine Komponente für sich, z.B. mit Unittests. Nachgelagerte Regressionstests sollen dann unerwünschte Nebenwirkungen von Änderungen in einer Komponente auf andere Teile des Systems aufdecken.

Wurden seit der letzten Integration viele Änderungen an einem Projekt vorgenommen, können während des Regressionstests viele Fehler auftreten, bei denen die Suche nach der Ursache eine erhebliche Herausforderung darstellt. Da wir es mit einer großen Menge an Codeänderungen in vielen Komponenten zu tun haben, ist die Problemdiagnose entsprechend schwierig.
Statt sich der Lösung echter Anforderungen zu widmen, ist das Entwicklungsteam damit beschäftigt, Fehler, die lediglich auf vorherigen Codeänderungen basieren, zu finden.

Die Problematik der Behebung von Integrationsfehlern wird im traditionellen Ansatz noch zusätzlich dadurch verstärkt, dass die Entwickler ihre Korrekturen nicht auf einem stabilen Build basieren lassen können. Wenn ein Fehler behoben wurde, können die Entwickler nicht sicher sein, dass andere fehlgeschlagene Regressionstests mit dem aktuellen Fix oder dem Berg von weiteren offenen Fehlern zusammenhängen.

Im schlimmsten Fall wird die Fertigstellung der Anwendung verlängert und gesetzte Termine können nicht eingehalten werden.

Warum sollte man kontinuierliche Integration verwenden?

Auf einen Nenner gebracht, ermöglicht kontinuierliche Integration die Entwicklung von besseren, stabileren Anwendungen bei gleichzeitiger Einsparung von Entwicklungskosten.

Dies hat auch zu einem entsprechenden Boom auf agile Softwareentwicklungsmethoden und kontinuierlicher Integration geführt.

Hierbei geht es nicht nur darum, einen Defekt in der Anwendung zu finden und zu korrigieren sondern auch darum weitere Defekte zu vermeiden. Je früher ein Defekt im Entwicklungszyklus gefunden und behoben wird, umso wahrscheinlicher ist es, dass weitere Defekte, die auf dem ursprünglichen basieren, ganz vermieden werden können.

Änderungen erfolgen immer basierend auf dem letzten, stabilen Stand der Anwendung im Versionierungssystem. Damit wird es viel einfacher die Ursache eines Fehlers zu identifizieren, sollte ein Test nach einem Build nicht mehr funktionieren.
Erweiterte Testfälle führen außerdem zu qualitativ besserem Code.
Zudem es ist auf jeden Fall besser, einen kurzen Test regelmäßig durchzuführen, als einen allumfassenden Testplan zu haben, der niemals ausgeführt wird.

Schließlich kann das Entwicklungsteam durch eine erhöhte Sichtbarkeit und Transparenz effizienter auf Probleme reagieren und hat außerdem noch eine bessere Sicht auf den aktuellen Gesamtzustand des Projekts.

Was ist kontinuierliche Integration

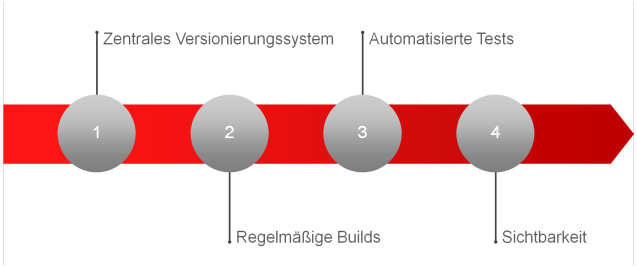

Abb. 1: Komponenten der kontinuierlichen Integration

Das Wort Integration in der Bezeichnung „kontinuierlicher Integration“ bezieht sich auf den Vorgang des Zusammenbauens (engl. Build) von Software in Vorbereitung auf die Verteilung dieser auf die Zielsysteme (engl. Deployment). Es ist nicht zu Verwechseln mit Integration im Sinne von Applikationsintegration, für das z.B. die Oracle SOA Suite verwendet wird.
Kontinuierliche Integration ist eine DevOps-Methode (DevOps = Development Operations; Partnerschaft zwischen Entwicklung und Betrieb), die eine Qualitätskontrolle bei der Herstellung von Software einführt. Ausgelöst werden kann diese durch Ereignisse wie „Nightly Builds“ oder – besser – jedes Mal beim Einchecken von Code in das Versionierungssystem.

Um kontinuierliche Integration zu erreichen, startet man mit einem *zentralen Versionierungssystem* wie CVS, Subversion oder Git. Dieses unterstützt die gemeinsame Entwicklung durch Änderungsverfolgung und Bereitstellung einer zentralen Informationsquelle für alle Projektdaten (engl. Single Source of truth).
Statt Dateien zwischen den Entwicklern direkt auszutauschen, werden alle Dateien auf kontrolliertem Weg über das Versionierungssystem geteilt. Ein Entwickler, der an Code arbeitet, den er aus dem zentralen Versionierungssystem bezogen hat, weiß das er auf dem neuesten Versionsstand der Software arbeitet.

Das Wort kontinuierlich in der Bezeichnung „kontinuierlicher Integration“ bezieht sich auf das *regelmäßige Build* der gesamten Anwendung. Dies kann an bestimmte Meilensteine oder Zeitpunkte gekoppelt sein, wie z.B. Nightly Builds. Im Falle von Nightly Builds wird der gesamte

Entwicklungsstand am Tagesende eingecheckt und der Build über Nacht ausgeführt. Am nächsten Morgen können die Entwickler anhand des Ergebnisses sehen, ob evtl. Fehler behoben werden müssen oder ob sie mit der Entwicklung, ausgehend von dem neuen, stabilen Versionsstand, weitermachen können.
Noch besser ist es allerdings, wenn ein Build jedes Mal ausgeführt wird, wenn Code in das zentrale Versionierungssystem eingecheckt wird. Hierdurch wird ein Buildtest auf dem kleinsten Delta an Codeänderungen erreicht. Sollte dabei ein Fehler auftreten, hat der Entwickler deutlich weniger Code zu analysieren, um diesen zu beheben.

Der erste intuitive Widerstand gegen kontinuierliche Integration basiert meistens auf der Erfahrung der Vergangenheit, wo die „Integration“ am Ende des Entwicklungszyklus eher schmerzhaft war und eine große Menge an Bugs zum Vorschein brachte. Dies ist aber genau das, was kontinuierliche Integration vermeiden will. Builds werden regelmäßig durchgeführt um Fehler früh zu finden und schnell zu beheben. Im Idealfall kann der Entwickler den Fehler direkt korrigieren, sollte ein Build nicht funktioniert haben.

Das Bauen der Software ist aber nur ein Teilaspekt. Testen ist eine weitere kritische Komponente. Um potentielle Regressionsfehler zu finden, die sich durch die Änderungen am Code eingeschlichen haben, sollten eine Reihe von *automatisierten Tests* nach jedem Build ausgeführt werden. Es sollte hierzu am besten eine Methode eingeführt werden, bei der die Entwickler die Tests mit den Änderungen liefern. Dies führt zu einer höheren Testabdeckung.

Schließlich ist auch noch eine gute *Sichtbarkeit* in die kontinuierliche Integration extrem wichtig. Wenn das Einchecken von Code zu einem fehlgeschlagenen Build führt, sollte das Entwicklungsteam direkt davon erfahren. Nun kann die Ursache zeitnah diagnostiziert und der Fehler behoben werden.

Der Entwicklungs- und Integrationszyklus

Die Hauptaufgabe eines Entwicklers besteht in der Entwicklung von Code. Hierbei kann es sich um neuen Code, um Erweiterungen an bestehendem Code oder auch um Korrekturen von Fehlern handeln.

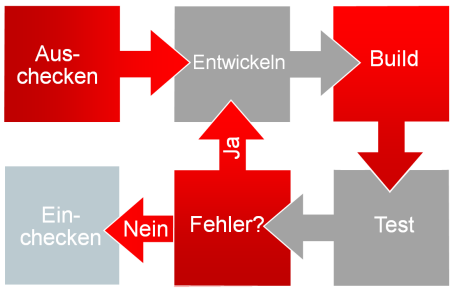

Abb. 2: Der Entwicklungszyklus

Der Entwickler baut oder kompiliert den Code und führt geeignete Tests durch, um etwaige Fehler zu finden. Wenn er Fehler findet, wird er den Zyklus wiederholen und die Fehler beheben, bis keine weiteren Fehler mehr auftreten. Den Abschluss bildet das Einchecken des Codes in das Versionierungssystem.

Nun ist die Zeit gekommen, um den Integrationszyklus durchzuführen: Der Buildprozess, der das Gesamtsystem baut bzw. integriert, wird ausgeführt. Dann werden weitergehende Tests gefahren, welche die Gesamtfunktionalität der Anwendung prüfen. Sollten einer oder mehrere dieser Tests

fehlschlagen, werden die Ursachen analysiert und die Defekte identifiziert. Nun können die Entwickler im Entwicklungszyklus die Defekte reparieren und der Integrationszyklus beginnt von vorne.

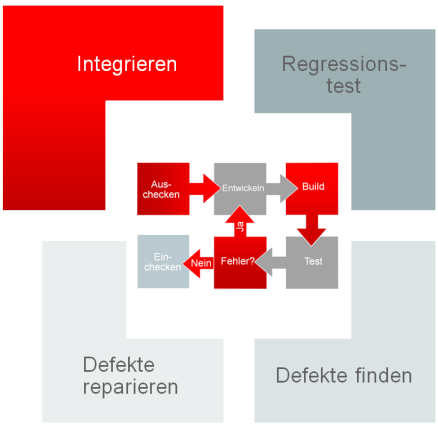

Abb. 3: Der Integrationszyklus

Kontinuierliche Integration mit Oracle SOA Suite in der Praxis

Der Oracle JDeveloper und die Oracle SOA Suite können unter Verwendung von weiteren Werkzeugen in einen automatisierten Integrationszyklus eingebunden werden.

Im Rahmen der Vertragsdemonstration wird als zentrales Versionierungssystem Subversion verwendet. Alternativ ist der Einsatz von Git möglich.
Als Buildsystem kommt Apache Maven zum Einsatz. Es können aber auch Ant oder selbstentwickelte Skripte verwendet werden.
Der CI Manager Hudson wird zur Automatisierung der Builds und Tests eingesetzt. Alternativen hierzu finden sich in Jenkins oder Cruise Control.

Die Demo wird den in Abb. 4 dargestellten Entwicklungszyklus zeigen.

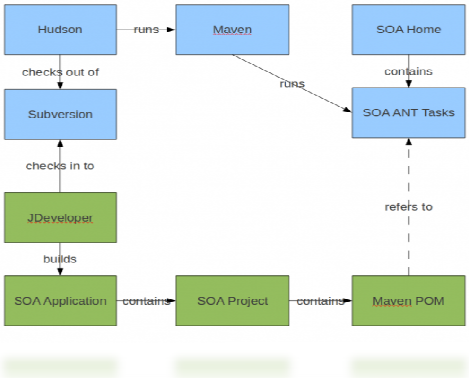

Abb. 4: Der Entwicklungszyklus der Demo

Die grünen Kästen illustrieren die Entwickleraktivitäten. In JDeveloper wird eine einfache SOA Anwendung entwickelt. Diese enthält ein SOA Projekt welches wiederum eine Maven POM (Project Object Model) Datei enthält, die den Build und das Deployment des Projekts beschreibt.

Die blauen Kästen zeigen, wo Hudson übernimmt. Den Startschuss für Hudson gibt das Einchecken der Änderungen in Subversion von JDeveloper aus.

Ausblick

Die Einführung von kontinuierlicher Integration in die Softwareentwicklung ermöglicht die Entwicklung von besseren, stabileren Anwendungen bei gleichzeitiger Einsparung von Entwicklungskosten.

Dem gegenüberstellen muss man die Einrichtung und Pflege eines Integrationsservers, wie er in diesem Artikel beschrieben wurde.

Im Rahmen seines PaaS Angebots bietet Oracle den Oracle Developer Cloud Service an, der die oben beschriebenen Funktionalitäten enthält. Weitere Details hierzu finden sie auf der Oracle Cloud Webseite unter https://cloud.oracle.com/developer.

Kontaktadresse:

Sören Halter
Oracle B.V. & Co. KG
Robert-Bosch-Str. 5

D- 65503 Dreieich

Telefon:	+49 (0) 261 6679441
Mobil:	+49 (0) 174 3443752
E-Mail	soeren.halter@oracle.com
Internet:	www.oracle.de