Dieser Artikel legt aus der Sicht des Managements die Gründe dar, warum sich in jedem größeren Softwareentwicklungsprojekt der Aufwand für ein umfassendes Continuous Integration (CI) lohnt.

Was ist CI?

CI ist die kontinuierliche Integration des Softwareprodukts entlang einer automatisierten Build-Pipeline. Praktisch heißt das, dass die Software mehrmals täglich mit der aktuellen Codebasis automatisch gebaut und umfangreichen automatisierten Qualitätsprüfungen unterzogen wird.

Das wichtigste Ergebnis ist dabei die Antwort auf die Frage, ob alle Regressionstests fehlerfrei durchlaufen wurden (grüne Regression), denn dann kann die getestete Version - natürlich abhängig vom Realisierungsstand - vom Testteam weiter untersucht oder sogar potentiell ausgeliefert werden.

Der Integrationsprozess wird vom zentralen Software-Repository (Version Control System - VCS) gespeist und auf einem oder mehreren dedizierten CI-Maschinen, also unabhängig von den Umgebungen der einzelnen Entwickler, ausgeführt.

Die Steuerung des automatisierten Ablaufs erfolgt durch einen **CI-Server**, der die nötige Funktionalität bereitstellt, um die Ausführung von Jobs zu steuern, Daten von einem Job an den nächsten weiterzugeben oder Daten zu einem Durchlauf zu sammeln und über entsprechende Reports zur Verfügung zu stellen. Ein CI-Server, der z.B. gerne im Java-Enterprise-Umfeld eingesetzt wird, ist Jenkins - ein Open-Source-Tool, für das eine Vielzahl von Plugins zur Verfügung steht.

Abhängig von den technischen Gegebenheiten des jeweiligen Projektes umfasst die Build-Pipline (siehe Abbildung 1) prinzipiell folgende Aufgaben:

- Build
- Database Integration
- Testing
- Inspection
- Feedback

Wird von Continuous Delivery gesprochen, kommen noch folgende Schritte dazu:

- Deployment
- Delivery

Die Anordnung der Test- und Inspektionsschritte ergibt sich aus der Komplexität der einzelnen Maßnahmen. Je aufwändiger die Durchführung ist, desto später wird sie ausgeführt, d.h. schlagen die einfacheren Maßnahmen bereits fehl, kann man sich die Ausführung der aufwändigeren Schritte möglicherweise sparen. Begonnen wird mit einem Smoke-Test, der die grundsätzliche Lauffähigkeit des Builds prüft. Dann folgen Regressionstests, deren vollständiger Durchlauf je nach Projektkomplexität durchaus mehrere Stunden dauern kann. Anschließend folgen Qualitätsmetriken und Lasttests. Abhängig von der Version des Builds werden zu guter Letzt gegebenenfalls noch systematische und explorative Tests durch das Testteam durchgeführt.

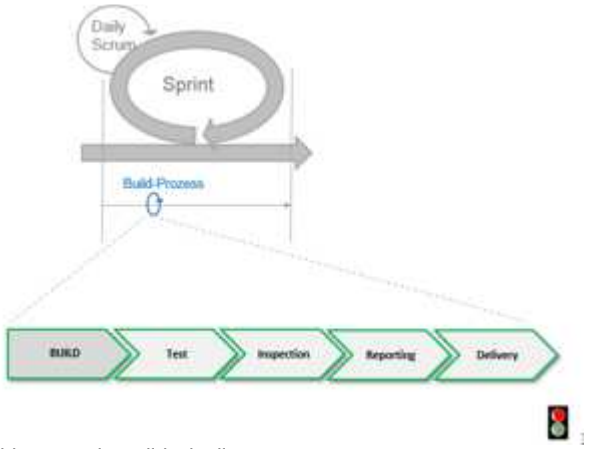

Abbildung 1: Die Build-Pipeline

Der Erstellungsprozess wird zeitnah zu den in das zentrale Repository eingecheckten Änderungen der Entwickler ausgeführt. Änderungen an der Code-Basis führen also dazu, dass das Produkt neu gebaut wird und somit geprüft wird, ob die aktuelle Codebasis noch „grün“ ist.

Folgendes ist dabei zu beachten:

- Die Integration wird auf dedizierten CI-Servern durchgeführt - nicht auf Entwicklermaschinen.
- Die Build-Pipeline ist automatisiert und wird nicht manuell gesteuert.

Ein wichtiges Ziel von CI ist ein schnelles Feedback über den Zustand der Software an die Entwickler, um potentielle Fehler möglichst schnell erkennen und beheben zu können. Das Feedback interessiert natürlich auch das Management, und so gibt es Projekte, bei denen anhand einer symbolischen Ampel für jedermann sichtbar angezeigt wird, ob die letzte Regression „grün“ oder „rot“ war.

Warum CI?

Im agilen Umfeld unabdingbar
Continuous Integration ist im agilen Umfeld - typischerweise in einem Scrum-Projekt - unabdingbar. Die kurzen Iterationszyklen machen es erforderlich, dass der Entwickler möglichst schnelles Feedback über den Zustand des von ihm eingecheckten Sourcecodes bekommt. Durch CI werden Integrationsprobleme so schnell wie möglich erkannt. Abbildung 2 veranschaulicht den Sachverhalt: Im Laufe eines Sprints wird die Software wiederholt neu gebaut und geprüft.

Objektivierung des Build-Prozesses
Durch CI wird der Integrationsprozess aus verschiedenen Gründen „objektiviert“:

- Der Integrationsprozess wird unabhängig von den Umgebungen der Entwicklermaschinen systematisch aufgesetzt; das Argument „aber auf meiner Maschine hat es doch funktioniert“ ist obsolet.

- Durch die Automatisierung lassen sich die erstellten Builds qualitativ miteinander vergleichen, damit lassen sich Kennzahlen definieren und Trends ableiten, die als Input zur Lenkung des Gesamtprozesses dienen können. Solche Kennzahlen können z.B. sein:
  - Qualitätsmetriken
  - Anzahl täglicher Commits
  - Anzahl täglicher Builds
  - Anzahl täglicher Builds mit „grüner Regression“

Aber Vorsicht bei den Kennzahlen. Diese müssen genau hinterfragt werden. Z.B. sagt die Zahl der täglichen Commits nur dann etwas über die Agilität aus, wenn die einzelnen Commits bestimmten Qualitätsanforderungen entsprechen müssen.

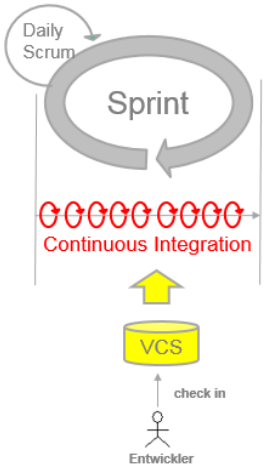

Abbildung 2: Einbettung CI in Scrum

Qualität steht ständig im Fokus
Durch Continuous Integration steht die Qualität ständig im Fokus aller Projektbeteiligten. Qualität wird nicht nachträglich hineingeprüft, sondern wird durch den Entwicklungsprozess gelebt.

Fehlende Automatisierungen sind technische Schulden
Folgendes führt u.a. im Zusammenhang mit CI zu technischen Schulden:

- unzureichende Automatisierung der Build-Pipeline,
- unzureichende Testabdeckung,
- ein zu hoher Anteil an manuellen Tests, obwohl eine Automatisierung machbar und sinnvoll wäre.

Aber gerade beim Abbau technischer Schulden ist CI unbedingt notwendig. Z.B. können Refactoring-Aktivitäten durch die gesamte Codebasis gehen - ohne Feedback durch CI können solche Aktivitäten ziemlich riskant sein.

Weniger Testen erhöht die Velocity
Dies ist ein Trugschluss, der immer dann bereitwillig gezogen wird, wenn keine Zeit mehr zum Testen bleibt. Aber diese Situation kann erst gar nicht entstehen, wenn CI und Test Driven Development (TDD) gemeinsam gelebt werden. Der Testaspekt ist dann keine separate Aktivität, die im Nachhinein möglicherweise noch ausgeführt wird, sondern der Entwickler denkt von Anfang an testfallorientiert; die ganze Vorgehensweise ist danach ausgelegt.

Das wichtigste Argument für CI aus Managementsicht
Die genannten Maßnahmen dienen dazu, die Risiken des Build-Prozesses zu minimieren, indem durch die Automatisierung der Prozess vereinheitlicht und objektiviert wird. CI gibt dem Management damit die Gewissheit, dass der Herstellungsprozess ausgereift ist. Letzten Endes wird damit das Vertrauen geschaffen, dass das Softwareprodukt jederzeit die festgelegten Qualitätsstandards erfüllt.

Häufig besteht bei Softwareentwicklungsprojekten ein Mangel an Transparenz gegenüber dem Management; das Management ist zu weit weg von der Technik und den Prozessen, um Zustand und Qualität hinreichend beurteilen zu können. CI hilft, den Mangel an Transparenz zu beseitigen.

CI gibt es nicht umsonst!

CI gibt es nicht umsonst. Zur Realisierung von CI muss einiges an Aufwand investiert werden; so sind u.a. folgende Maßnahmen durchzuführen:

- Commitment des Teams einholen,
- den Softwareentwicklungsprozess entsprechend auslegen,
- Build-Prozess (CI-Prozess) definieren und umsetzen,
- Einrichten einer zentralen Codebasis,
- CI-Server einrichten, konfigurieren,
- Testumgebungen einrichten, konfigurieren,
- Metriken definieren, abgreifen und analysieren,
- CI-Best-Practices umsetzen,
- …

Fazit

Nicht nur das Entwicklungsteam, sondern gerade auch das Management sollte daran interessiert sein, durch Continuous Integration ein Werkzeug an die Hand zu bekommen, durch das

- die Build-Prozesse automatisiert werden,
- Risiken im Softwareentwicklungsprozess reduziert werden,
- wichtige Kennzahlen über das Produkt und die Prozesse zur Verfügung gestellt werden.

Es ist ein Trugschluss, zu glauben, dass man Zeit spart, wenn man auf die Einführung von CI im Projekt verzichtet. Aber: CI ist kein Allheilmittel sondern ein Element einer umfassenden Projektphilosophie, die noch andere Aspekte wie z.B. Teamleistung, Test Driven Development etc. umfassen muss.