Titel

Wir sind 4 Architekten und haben letztes Jahr zusammen eine Workshopreihe zum Thema Buildsysteme durchgeführt

Was erwartet uns?

Um Buildsysteme zu bewerten wollen wir zuerst unsere Erwartungshaltung formulieren. Dann kommen wir natürlich nicht ohne die Platzhirsche

Ant und Maven aus – wir schaun uns die beiden Kandidaten an und versuchen Sie einzuordnen

Zu guter Letzt denken wir darüber nach, wie win Buildsystem aussehen könnten, wenn wir uns eines wünschen dürften :-)

Erwartungshaltung an Buildsysteme

Zentrale Forderung CRISP

- Complete (das Buildsystem steht als Paket zur Verfügung)
- Repeatable (bei Widerholten builds identische Artefakte)
- Informative (automatisierte Tests, Reports)
- Schedulable
- Portable (Windows, *nix, Mobile, usw.)
- Erwartungshaltung an Buildsysteme

Darüber hinaus:

- Multiprojektfähigkeit: gute Unterstützung für builds mit vielen Projekten (Abhängigkeiten sind modular beschreibbar, das Projekt ist in Teilen baubar)
- IDE-Integration nach dem DRY-Prinzip (dont repeate yourself) in Eclipse, Idea, Netbeans, ...
- performant (um minimale Roundtrip-Zeiten zu ermöglichen)
- 3rd Party-Lib Verwaltung

Darüber hinaus:

- Reifegrad: stabile Releases – stabile Tools – Konzepte durchgängig funktional
- einfach und angemessen (essential, not accidental complexity, Lernkurve)
- Convention over Configuration: Gute Konventionen als Voreinstellung
- leicht anpassbar bei abweichenden Anforderungen

Erstes Beispiel: Ant

Ant ist eine umfassende Sammlung von Build-Werkzeugen, d. h.

- alles Wichtige vorhanden
- deklarative Beschreibung der Buildreihenfolge
- Dependecies nur zwischen Targets

Einordnung von Ant

- Complete:
  - Wichtige Tools wie Compiler, Packer und Fileoperationen sind im Releaseumfang vorhanden
  - Mächtige Basis Mechanismen für den Umgang mit Dateien, Verzeichnisse und Archiven

- Repeatable: beim nochmaligen Bauen sind gleiche Ergebnisse gut erreichbar.
- Informative: Hier kommt die erste ernsthafte Schwäche von Ant zum tragen – Reports für die Testergebnisse, Codequalität, Testabdeckung usw. sind zwar als Erweiterungen vorhanden. Ein integriertes zusammen ist aber nur sehr schwer erreichbar.
- Portable: Überall, wo java läuft
- Hoher Reifegrad: Schon lange im Einsatz, daher Erweiterungen für alle Lebenslagen verfügbar
- Multiprojektfähigkeit sind out of the Box nicht unterstützt (Abhilfe: z.B. Ivy)
- Integration in alle relevanten IDEs verfügbar aber nicht vollständig (multiprojekt)
- Performance ganz okay – verteilt aber nicht so ohne weiteres
- Verwaltung von 3rd Party-Libs fehlt vollständig
- Konventionen gibt es nur sehr unzureichen (Targets Clean, ...)
- Anpassbarkeit: Ist fast alles möglich, aber nicht immer einfach

Erfahrung mit ANT

- Manchmal ist Imperative Verarbeitung wünschenswert (Reihenfolgen, Schleifen ...)
- Builddefinition in XML ist gesprächig
- build.xml als Sprache verstanden hat deutliche Lücken – sobald Sie mal versuchen, ein fremdes Buildscript zu lesen oder eine Klasse von ähnlichen Tasks zu schreiben werden Sie das sehr schnell merken.

Zweites Beispiel: Maven

Wenn Ant eine Werkzeugkiste ist, dann ist Maven ein vollständiges Buildsystem. Zusätzlich zu der Werkzeugfülle von Ant bietet Maven:

- Multiprojekt Support – hierarchisch oder flach
- Ein 3rd Party Bibliothekskonzept, in dem mehrfach genutzte Libraries endlich ordentlich verwaltet werden können (log4j)
- Mut zu meist sinnvollen Konventionen – Verzeichnisstruktur
- Integriertes Reporting, das recht schnell aufgesetzt werden kann – aber nur im flachen Multiprojekt
- Es gibt Scopes für Compile, Test, Packaging

Einordnung von Maven

- Complete: ja – ist alles essenzielle vorhanden – leider nicht immer als teil eines stabilen Releases
- Repeatable: Ganz großes Manko! - 3rd Party Libs und Maven-Module werden transparent nachgeladen, d.h. zwei Builds mit selbem Stand zu unterschiedlichen Zeiten können unterschiedliche Ergebnisse liefern – das passiert sogar recht häufig.
- Informative: Die Integration der gängigen Reports ist ganz okay, bei weitergehenden Konfigurationswünschen wird aber schnell die unterschiedliche Reife der einzelnen Module spürbar.
- Portable: Überall, wo java läuft und wo ein Netzzugang vorhanden ist (transparentes Nachladen)
- Hoher Reifegrad: Benötigte Module sind häufig nur in einer Entwicklerversion verfügbar – entsprechend oft gibt’s Instabilitäten.

- Multiprojektfähigkeit sind out of the Box unterstützt
- Integration in alle relevanten IDEs verfügbar aber nicht immer dry
- Performance ist schlecht
- Verwaltung von 3rd Party-Libs ist enthalten aber oft umständlich – viele kleine Versionen, auf die Maven Bibliothek festgelegt, keine richtige Wahl über die Granularität, komplexe Repository Struktur
- Konventionen sind meist richtig klasse – nur gibt's keine effektive Möglichkeit davon abzuweichen – manchmal etwas overengeneered
- Anpassbarkeit: Entgegen der Konventionen und weitgehenden Konzepte schwierig

Erfahrungen mit Maven

- Wegen dem transparenten Netzzugang für libs und Module ist ein Arbeiten ohne Netzzugang fast unmöglich. Es gibt zwar Möglichkeiten, man muss dazu aber Proxy- Und Spiegelrepositories aufsetzen und verwalten.
- Transitivität im Compile-Scope verursacht intransparente Abhängigkeiten zu 3rd Party Libs (Wissen Sie, von wieviel anderen Projekten Spring, Hibernate oder Seam abhängen?)
- Die Einschränkung auf ein Artefact pro Projekt bläht den Build künstlich auf – ist echt nervig, wenn Sie Funktionalität in einem War Projekt in einem weiteren Projekt integrativ testen wollen ...
- Viele Maven Module funktionieren nicht sauber im Multiprojekt Modus – nur flat oder nur hierarchisch
- Keine Unterscheidung zw. Projekt und 3rd Party Libs in der Abhängigkeits def – damit gibt's sofort eine semantische Lücke zu Eclipse (Grenzen von Refactorings oder dem bearbeiteten Projekt)
- Unterschiedlicher Reifegrad der Maven Module – macht sich in unterschiedlichen Konfigurationsphilosophien bemerkbar – oftmals Zwang, das neueste Entwicklermodul zu verwenden – mit den entsprechenden Folgen für die Stabilität.

Fragen, die offen bleiben

Warum XML und weitere ConfigFiles als Buildsprache?
Gibt es einen angemessenen Mittelweg in der Bibliotheksverwaltung?

Buildng – was ist drin

kein XML – jeder build ist ein Java-Programm
nutzt ANT – bietet ein fluent interface für Ant in Java an (elegAnt)
unterscheidet Abhängigkeiten zu anderen Projekten und zu Bibliotheken (jars)
keine Versionen von Bibliotheken
Bibliotheken liegen im Dateisystem, nicht in irgendeinem Repository. Keine Vorgaben zur Verzeichnisstruktur, keine Verwaltung von Versionen
Compile-Abhängigkeiten sind nicht-transitiv, Test/Runtime-Abhängigkeiten sind transitiv
sinnvolle Konventionen (a la Maven)
Buildng – was fehlt noch
Infrastruktur für Berichte
builder für ejb-module, ear
Ide Integration