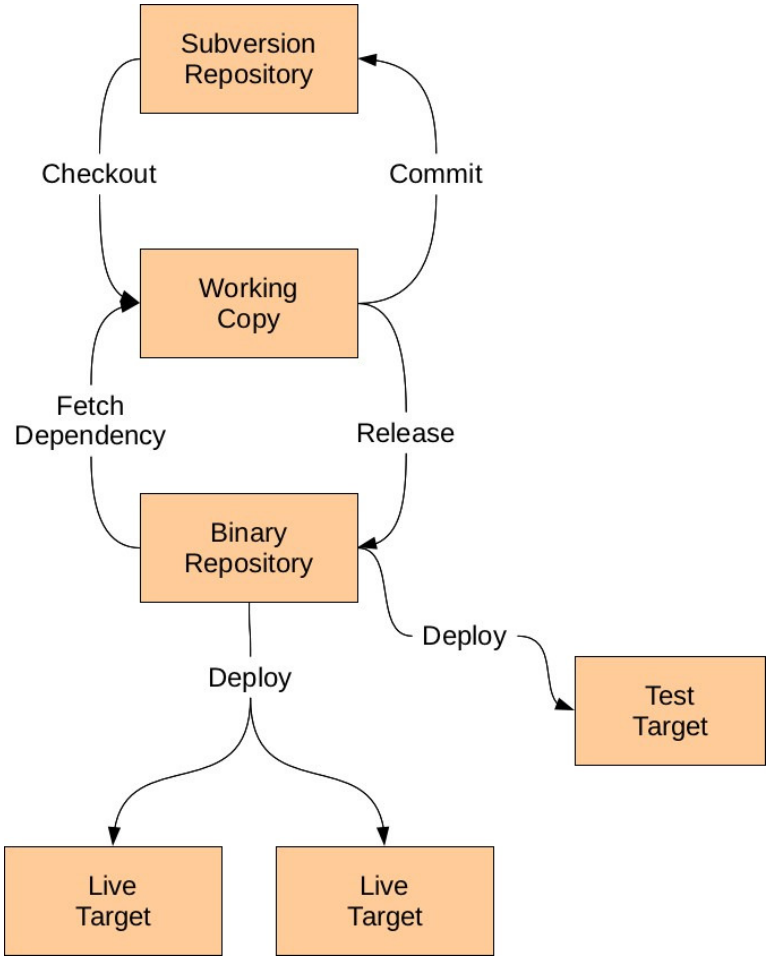
Buildsystem

Maven & Scons

Controls Entwicklungsforum

Januar 2012

a call from the past



Binary Repository

- Speichern von Artefakten (z.B. Shared Library und zugehörige Header)
- Versionierung von Artefakten
- Eindeutige Benennung von Artefakten (groupid, artifactid, version)
- Austausch von Artefakten
- Maven Repository layout ist der am weitesten verbreitete Standard

Maven

- Build-Management Tool
  - kompilieren, paketieren, verteilen
- Geschrieben in Java, für Java
- “Makefile” pom.xml
  - XML, beschreibend
- Ergebnis sind Artefakte
  - Jar, Zip, Tar, etc.

Maven Lifecycle

• Validate	überprüfe pom.xml
• Generate-sources	run precompilers
• Compile	erzeuge Objekte
• Test	run Unit-Tests
• Package	Archiv der Objekte erzeugen
• Install	in privates Binär-Repository
• Deploy	in gemeinsames Binär-Repository

Maven Dependencies

- Deklaration von Abhängigkeiten
  - groupid, artifactid, version(-range)
  - `de.gsi.bel.frontend : vmedevice_ppc : 1.0.0`
- Auflösung durch suchen in
  - Projekt (wenn multi-artifact)
  - privatem binär-Repository
  - gemeinsamen binär-Repository
- Auflösen transienter Abhängigkeiten

Maven + SCons

Maven kann mit nativem Code nicht brauchbar umgehen.
Daher:

- Maven generiert Konfigurationsdateien für SCons
- SCons übernimmt die Compile-Aufgabe des Lifecycles
- Alles weitere erledigt Maven.

Konfiguration

- Konventionen (z.B. src Folder)
- Mavens pom.xml
- Scons SConscript

Hello World - Tree

C:

```
-- helloworld
    |-- pom.xml
    +-- src
        |-- SConscript
        +-- helloworld.cc
```

Java:

```
-- helloworld
    |-- pom.xml
    +-- src
        |-- java
            +-- HelloWorld.java
```

Hello World - SConscript

```
Import('env')
SOURCES = env.Glob('*.cc')
prog = env.Program(
   'helloworld', SOURCES)
env.InstallBin(prog)
```

Hello World – pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>de.gsi.bel.frontend</groupId>
  <artifactId>helloworld</artifactId>
  <version>1.0.0</version>
  <packaging>scons</packaging>

</project>
```

Hello World – pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.gsi.bel.frontend</groupId>
  <artifactId>helloworld</artifactId>
  <version>1.0.0</version>
  <packaging>scons</packaging>

  <properties>
    <scons.arch>amd64</scons.arch>
  </properties>
</project>
```

Hello World – pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.gsi.bel.frontend</groupId>
  <artifactId>helloworld</artifactId>
  <version>1.0.0</version>
  <packaging>scons</packaging>
  <properties>
    <scons.arch>amd64</scons.arch>
  </properties>
  <build>
      <plugins>
          <plugin>
              <groupId>de.gsi.bel.maven.plugins</groupId>
              <artifactId>maven-scons-plugin</artifactId>
              <version>1.0.2</version>
              <extensions>true</extensions>
          </plugin>
      </plugins>
  </build>
</projcect>
```

Hello World - run

- `mvn validate`
- `mvn generate-sources` → target verzeichnis
- `mvn compile` → build und install verzeichnis
- `mvn package` → artifact
- `mvn install` → lokales repository
- `mvn clean`

Dependency

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>de.gsi.bel.chandel.hello</groupId>
      <artifactId>header</artifactId>
      <version>1.0.0</version>
      <type>zip</type>
    </dependency>
  </dependencies>
</project>
```

Dependency Beispiel

```
de.gsi.bel.frontend:nativedevice_amd64:scons:1.0.1
+- de.gsi.bel.frontend:accdevice_amd64:zip:1.0.1
|  +- de.gsi.bel.frontend:usrs_amd64:zip:1.0.1
|  |  \- de.gsi.bel.frontend:accdata_amd64:zip:1.0.1
|  |     \- de.gsi.bel.frontend:corbaifc_amd64:zip:1.0.1
|  +- de.gsi.bel.frontend:alarm_amd64:zip:1.0.1
|  |  +- de.gsi.bel.frontend:os_amd64:zip:1.0.1
|  |  |  +- de.gsi.bel.frontend:header_noarch:zip:1.0.1
|  |  |  \- de.gsi.bel.system:omniorb:zip:meta:4.1.3
|  |  \- de.gsi.bel.system:boost:zip:meta:1.41.0
|  \- de.gsi.bel.frontend.nameservice:tcpip_amd64:zip:1.0.1
+- de.gsi.bel.scons:omniidl:zip:1.0.0
\- de.gsi.bel.scons:gsimsgc:zip:1.0.0
```

Multimodule

```
<project>
  …
  <modules>
    <module>header</module>
    <module>program</module>
  </modules>
</project>
```

Vorteile

- Keine Umgebungsvariablen
- Abhängigkeiten
  - Deklaration
  - Auflösen
- Artefakte
  - versioniert
  - auffindbar, austauschbar
- Automatisierbar → Nightly build
- Plugins

Nachteile

- XML
  - ausführlich
  - wiederholend
- Leider immer noch Systemabhängigkeiten
  - Oracle
  - Crosscompiler
- Komplex
  - besonders im Fehlerfall

Multiarch

- Für jede Architektur ein Modul

```
-- helloworld
  |-- pom.xml
  +-- amd64
       +-- pom.xml
  +-- ppc
       +-- pom.xml
  +-- src
       +-- SConscript
```

Refresh SCons (1)

- Software Construction Tool
- Ersatz für Make
- Unabhängig von System-Umgebungsvariablen
  - Explizite Konfiguration
  - Reproduzierbarkeit
- “Makefiles” sind Pythoncode

Refresh SCons (2)

- Verwendung von Buildern statt Programmaufrufen
  - `SharedLibrary`
  - `Programm`
- Programmierbar
  - z.B. `Msgc` Builder
- Automatische Code Dependency
  - ObjectB benötigt ObjectA