Bauen mit Tycho

Mirko Swillus
Qualitype AG
18.10.2012, JUG Saxony

Bauen mit Tycho

1. Vorstellung und Motivation
2. Bauen
3. Laufzeitcontainer: Equinox
4. Repositories
5. Testen
6. Schwierigkeiten und Lösungen
7. Ausblick

Qualitype AG

- Gründung 2001, Sitz in Dresden Hellerau
- 30 Mitarbeiter, 20 Entwickler
- Forensik, Klinik, Lebensmittel
- Produkte und Entwicklungsdienstleistungen
- Technologie (unter anderen): Eclipse RCP, J2EE mit jBoss AS
- www.qualitype.de

Mirko Swillus

- Diplom Informatiker (BA)
- Entwickler bei der Qualitype AG seit 2008
- Interessen: Eclipse RCP, J2EE, Open Source und offene Standards
- Release Engineering und Build Management
- twitter.com/mechko

Tycho auf einer Folie

- Satz von Maven Plug-Ins, erlaubt das Bauen von Plug-Ins, Features, Produkten
- „Manifest First“
- Ursprünglich entwickelt von Sonatype Inc. (neben SAP Hauptkontributor)
- Seit 2011 offizielles Eclipse Projekt, Incubation Status
- Gebaut mit Tycho: EGit, m2eclipse, CDT, JBoss Tools
- Aktive Community, Liste tycho-user@eclipse.org
- www.eclipse.org/tycho

Motivation für Apache Maven

- Exkurs: Apache Maven
  - Weit verbreitetes, populäres Java Buildtool
  - Jedes Projekt erhält eine POM (Project Object Model)
  - M2-Repositories für Binary- und Sourceartefakte (Maven Central)
  - Build wird nachvollziehbar und reproduzierbar
  - Kommandozeilentool, ideal für Headless Builds:
    `git clone <something>; mvn clean install`
  - Plug-In Konzept, Vielzahl von Community Plug-Ins (z.B. Apache Sonar)
  - Eclipse-Integration über m2e (http://eclipse.org/m2e)

Motivation für Eclipse Tycho

- Schwächen durch Ant-basierten PDE Build
- Repository Management (Infrastruktur mitunter vorhanden)
- Headless Produkte bauen
- UI-Tests integrieren
- Codequalität messen
- Continious Integration, zentraler Buildserver

PDE Build Folklore

- Projektspezifische Build Environments
  - enthalten alle Requirements als Kopie (enorme Redundanz)
  - definiert die Target Platform
- Vielzahl von Requirements im lib-Folder (Eigene Libs, Third-Party)
  - Abhängigkeiten von Libraries wurden nie ausgewertet!
- Jeder Entwickler baut lokal auf seiner Maschine:
  - nicht standardisiert, „individuelle“ Ergebnisse
  - nicht nachvollziehbar
- Hoher Aufwand durch manuelles Kopieren, bzw. durch Pflege von Ant-Skripten

Bauen.

– Packaging Types – Target Provisioning – Parents –

Tycho Bauwerke

- Packaging Types:
  - eclipse-plugin (RCP Plug-Ins, Patch Fragmente)
  - eclipse-feature (RCP Features)
  - eclipse-repository (Produkte)
  - eclipse-test-plugin (Integration Tests)
- Gewöhnungsbedürftig - Jedes Artefakt hat zwei Versionen:
  - Eclipse-Welt: Bundle (MANIFEST.MF), Feature, Product-Version (XMLs)
  - Maven-Welt: Project Version (oder Parent Version)
  - Feature-Request anhängig, aber offensichtlich schwierig

Target Provisioning

- Jeder Build braucht eine Target Platform
- Alle Requirements müssen enthalten sein (ID, Version)
- In Tycho zwei Möglichkeiten:
  - .target-File: Repositories und konkrete IUs (Installable Units)
  - Allgemeine <repository> Definition in der pom.xml (Layout p2)

Das Targetfile

- Duale Nutzung durch PDE (Entwicklung) und Tycho (Build)
  - PDE: Target File Editor (mit Link „Set as Target Platform“)
  - Tycho: target-platform-configuration
- Vorteile
  - Probleme mit Requirements sofort in der IDE erkennen
  - Sehr granulare Kontrolle über die Target Platform
  - Trennung in Release-Target und Snapshot-Target
- Nachteile
  - Jede Änderung muss manuell (!) im Targetfile gepflegt werden
  - Interpretation des Targetfiles von Tycho und PDE nicht symmetrisch (0.0.0)

Der Uberparent

- Bewährt: Ein allgemeiner RCP Parent
- Packaging: Pom
- Definiert
  - Tycho-Version
  - Aktiviert Tycho-Source-Plugin
  - Pattern für Build Qualifier
  - Gegen welches Targetfile gebaut werden soll (Profiles)

Build Parents

- Sinnvolle Zusammenfassung von Projekten unter einem Build Parent
- Werden zusammen in einem Reaktor gebaut
- Alle Projekte darunter werden als Maven Module aufgenommen
- Gemeinsame Versionsnummer (geerbt vom Parent)
- Populäre Szenarien:
  - Mehrere Plug-Ins und ein Feature, was alle Plug-Ins enthält
  - Ein Feature, ein Produkt und ein Test-Plug-In

Ein Name für alles

- Bundle-SymbolicName: com.qualitype.example.texteditor
- Maven ArtifactId: com.qualitype.example.texteditor
- Folder im Buildprojekt (auch SCM): com.qualitype.example.texteditor
- .project Name: com.qualitype.example.texteditor

Mehr ist es oft nicht.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <artifactId>com.qualitype.rcp.example.texteditor</artifactId>

     <parent>
          <groupId>com.qualitype.rcp</groupId>
          <artifactId>com.qualitype.rcp.example.build</artifactId>
          <version>2.0.2-SNAPSHOT</version>
     </parent>

     <packaging>eclipse-plugin</packaging>

</project>
```

Wofür wird gebaut?

Equinox!

- OSGi und Equinox - Versionierung - Libraries im Container -

OSGi und Equinox

- Equinox: Eclipse Projekt, Implementation des OSGi-Standards
- Wenige Erweiterungen, einige spezifische Vorlieben
  - Require-Bundle (Equinox), Imported-Packages (OSGi)
- Bringt eigenes Provisioningsystem mit: Equinox p2

Versionierung

- Maven Konzept: Releases und Snapshots (getrennte Repositories)
- Maven: 1.2.3-SNAPSHOT < 1.2.3
- OSGi: 1.2.3.qualifier > 1.2.3
- Lösung per Konvention:
  Snapshots **und** Releases erhalten eindeutigen Build Identifier:
  - Snapshot: 1.2.3.SNAPSHOT-201210181900
  - Release: 1.2.3.v201210181900
  - Snapshots immer kleiner als Release
- OSGi Draft schlägt negative Qualifier für Snapshots vor:
  1.2.3-201210181900 < 1.2.3.201210181900

Library als OSGi-Bundle

- Früher: Build durch Ant, Kopie in den lib-Folder
- Jetzt: Build durch Maven, Deployment ins Repository
  - maven-bundle-plugin (Apache Felix)
- Konfiguration, welche Packages exportiert werden sollen
- Import-Packages werden berechnet

Third Party Libraries

- Einige Libraries leider noch nicht OSGi-sified
- Alternative: Selber mit eigenem Bundle wrappen
- Öffentliche OSGi Repositories:
  - Eclipse Orbit: http://www.eclipse.org/orbit/
  - Springsource: http://ebr.springsource.com

Das Repository ist alles.

– Sonatype Nexus – Equinox p2 – Nexus p2 Plug-Ins – p2 Mirrors – – Tooling –

Maven Repositories

Sonatype Nexus – Repository Management Server:

- ✓ Standard m2-Repositories plus Weboberfläche
- ✓ Differenzierte Lese- und Schreibrechte
- ✓ Läuft in einem Jetty (kommt mit)
- ✓ Nexus OSS mit Open Source Lizenz (AGPL)
- ✓ Proxy für entfernte Repositories (Maven Central)
- ✓ Einfache Integration in lokales Firmennetzwerk

Sonatype Nexus Maven Re

/nexus/index.html#view-repositories;qualitype_rcp~browsestorage

g+ | Java | mko | qt rel p2 | qt snap p2 | qt central p2 | qt rcp p2 | qt rcp snap p2 | QT | ST | Themen | Client Build | wall

Qualitype

mirkos | Log Out

Sonatype Nexus™ Open Source Edition, Version: 2.0

Sonatype™ Servers

Nexus

Artifact Search

Advanced Search

Views/Repositories

- Repositories
- Repository Targets
- Routing
- System Feeds

Security

- LDAP Configuration
- Privileges
- Roles
- Users

Administration

- Capabilities
- Log Configuration
- Plugin Console
- Scheduled Tasks
- Server
- System Files

Help

Welcome | **Repositories**

Refresh | Add... | Delete | Trash... | User Managed Repositories

Repository	Type	Quality	Format	Policy	Repository Status	Repository ...
Qualitype Snapshots	hosted	UNAVAILABLE	maven2	Snapshot	In Service	https://sourc...
RCP Qualitype	hosted	UNAVAILABLE	maven2	Release	In Service	https://sourc...
RCP Qualitype Snapshots	hosted	UNAVAILABLE	maven2	Snapshot	In Service	https://sourc...

RCP Qualitype

Browse Storage | Browse Index | Configuration | Mirrors | Summary | Artifact Upload

Refresh | Path Lookup:

- rap
- rcp
  - com.qualitype.rap.commons.feature
  - com.qualitype.rcp.alien
  - com.qualitype.rcp.apache.poi
  - com.qualitype.rcp.canvas
  - com.qualitype.rcp.chimerism
  - com.qualitype.rcp.chimerism.report
  - com.qualitype.rcp.commons
    - 4.0.0
      - com.qualitype.rcp.commons-4.0.0-p2artifacts.xml
      - com.qualitype.rcp.commons-4.0.0-p2artifacts.xml.md
      - com.qualitype.rcp.commons-4.0.0-p2artifacts.xml.sha
      - com.qualitype.rcp.commons-4.0.0-p2metadata.xml
      - com.qualitype.rcp.commons-4.0.0-p2metadata.xml.m
      - com.qualitype.rcp.commons-4.0.0-p2metadata.xml.sl

Maven Information | **Artifact Information**

Repository Path:	/com/qualitype/rcp/com.qualitype.rcp.com 4.0.0.jar
Uploaded by:	mirkos
Size:	694.97 KB
Uploaded Date:	Tue Apr 24 2012 10:00:38 GMT+0200 (Mitteleuropäische Sommerzeit)
Last Modified:	Tue Apr 24 2012 10:00:38 GMT+0200 (Mitteleuropäische Sommerzeit)

Download | Delete

Checksums

SHA1	d838a7a288f8e59f05a096ac7999c1b

Equinox p2 Repositories

- Content- und Artifacts-Repository
- content.xml
  - Installable Units (IU) mit Id und Version
    - Provides, Requires
    - Namespaces: osgi.bundle, java.package, ...
- artifacts.xml
  - Sizes, Hashes, RepositoryPath
- Generierung der Metadaten über PDE oder Tycho
- Keine eigene p2-Repository Management Lösung durch Eclipse

nexus-p2-repository-plugin

- Von Sonatype, verfügbar für Nexus OSS (und Professional)
- ✓ Konfiguration für jedes m2-Repository möglich
- ✓ Generiert für jedes deployte OSGi-Bundle p2-Metadaten
- ✓ Metadaten werden anschließend in ein eigenes p2-Repository eingefügt
- ✓ Jedes so konfigurierte m2- erhält ein zugehöriges p2-Repository
- Funktioniert nicht für Eclipse Features

nexus-p2-tycho-aggregator-plugin

- Eigenes Plug-In, Motivation: Eclipse Features verwalten
- GitHub: https://github.com/mechko/nexus-p2-tycho-aggregator-plugin
  - ✓ Fügt die von Tycho generierten Metadaten in das p2-Repository ein
  - ✓ Funktioniert damit auch für Eclipse Features
- Bekannte Probleme:
  - Source-Bundles werden im Repository falsch verlinkt
  - Temporäre Dateien müssen per Cronjob aufgeräumt werden (Nexus Bug anhängig)
  - Code-Qualität aus der Kategorie: Quick Hack
- Alternative: Patch für nexus-p2-repository-plugin, um Features zu unterstützen (Pull-Request anhängig)

Repository Best Practice

- Interne m2 Repositories bei der Qualitype (alles Nexus):
  - Libraries und Server Komponenten:
    - qualitype und qualitype_snaphots
    - nexus-p2-repository-plugin (für OSGi Bundles)
  - RCP Plug-Ins und RCP Features
    - qualitype_rcp und qualitype_rcp_snapshots
    - nexus-p2-tycho-aggreator-plugin (für Plug-Ins und Features)

p2 Mirrors

- Anforderung: Alle Artefakte sollen zentral verwaltet werden
- Problem: Entfernte p2 Repositories mitunter schlechte Performance (download.eclipse.org), Bandbreite
- Lösung: Lokale, firmeninterne p2 Mirrors
- Vereinfachung: https://github.com/briandealwis/p2-scripts

UnitsAndVersions.js

- Problem: Auffinden von Artefakten bei mehreren p2 Repositories schwierig
- Lösung: Eigenes Tool (in node.js)
  - ✓ Durchsucht mehrere lokale p2 Repositories nach einer IU (Installable Unit)
  - ✓ Zeigt geordnet alle verfügbaren Versionen für diese IU
  - ✓ Zeigt Requirements für diese IU
  - ✓ Kann beschränkt werden auf „Nur Release Repositories“
- Bei Intresse: Cloning auf GitHub

Largest version of ***com.qualitype.rcp.commons.feature.feature.group*** **(23 found)** Show Releases Only

Expand All | Collapse All

- **2.1.2.v201210121441** (qualitype_rcp) ©
  - com.qualitype.rcp.commons [4.1.4.v201210121432,4.1.4.v201210121432]
  - com.qualitype.commons.qtcommons [5.1.14.v201209271027,5.1.14.v201209271027]
  - org.apache.commons.beanutils [1.8.3,1.8.3]
  - org.apache.commons.io [2.3.0,2.3.0]
  - org.apache.commons.codec [1.6.0,1.6.0]
  - org.apache.commons.collections [3.2.1,3.2.1]
  - org.apache.commons.lang3 [3.1.0,3.1.0]
  - org.apache.commons.math [2.2.0,2.2.0]
  - org.apache.commons.logging [1.1.1.v201101211721,1.1.1.v201101211721]
  - com.springsource.javassist [3.15.0.GA,3.15.0.GA]
  - net.sf.paperclips [1.0.4.200908120926,1.0.4.200908120926]
  - net.sf.paperclips.ui [1.0.4.200908120926,1.0.4.200908120926]
  - com.qualitype.rcp.print [2.0.3.v201208311631,2.0.3.v201208311631]
  - com.qualitype.rcp.commons.rcp [4.1.4.v201210121432,4.1.4.v201210121432]
  - com.qualitype.rcp.jface [1.0.3.v201205071308,1.0.3.v201205071308]
  - com.qualitype.commons.qtejb [3.0.5.v201205071430,3.0.5.v201205071430]
  - com.qualitype.rcp.commons.feature.feature.jar [2.1.2.v201210121441,2.1.2.v201210121441]
- **2.1.2.SNAPSHOT-201210090818** (qualitype_rcp_snapshots) ©
- **2.1.1.v201209181113** (qualitype_rcp) ©
- **2.1.1.SNAPSHOT-201209071558** (qualitype_rcp_snapshots) ©
- **2.1.0.v201208311616** (qualitype_rcp) ©
- **2.0.11.v201208301356** (qualitype_rcp) ©

Testen.

- SWTBot - Continious Integration -

SWTBot

- Eclipse Projekt (Incubation): http://eclipse.org/swtbot
- Bietet Java API, um einfache funktionale Tests zu schreiben
- Ausführung über
  - IDE (eigene Run-Configuration)
  - Tycho (eclipse-test-plugin, läuft als Integration Test)
- Test-Plug-In benötigt Abhängigkeiten auf SWTBot-Bundles
- Über Mavens toolchain-Mechanismus muss die richtige JRE für den Test gesetzt werden

Testszenarien

- Alleinstehendes Plug-In testen:
  - Eigene Testapplikation schreiben
  - Funktionen des Plug-Ins werden dort über eine UI zugänglich gemacht
  - SWTBot-Tests testen die Funktionen
- Produkt testen:
  - Build-Parent des Produkts enthält eclipse-test-plugin
  - Enthält nur die Testklassen und benötigte Abhängigkeiten (auch auf SWTBot)

Aboutbox testen

```java
package com.qualitype.rcp.example.test;

import static org.junit.Assert.assertTrue;

import org.eclipse.swtbot.eclipse.finder.SWTWorkbenchBot;
import org.eclipse.swtbot.swt.finder.exceptions.WidgetNotFoundException;
import org.eclipse.swtbot.swt.finder.junit.SWTBotJunit4ClassRunner;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * Tests if some simple product configurations has been applied.
 *
 * @author Mirko Swillus
 */
@RunWith(SWTBotJunit4ClassRunner.class)
public class ApplicationTest {

	SWTWorkbenchBot bot;

	@Before
	public void setup() {
		this.bot = new SWTWorkbenchBot();
	}

	@Test
	public void testAboutText() throws Exception {
		this.bot.resetWorkbench();

		this.bot.menu("?").menu("Über").click();
		assertTrue("Could not find about box.",
				this.bot.activeShell().getText().equals("Info über Abetter LIMS"));
		assertTrue("Could not find content of about box.",
				this.bot.activeShell().bot().styledText().getTextOnLine(0)
				.contains("Abetter LIMS"));
	}
}
```

Continious Integration

- Genutzte Lösung: Jenkins Integration Server
- Workflow Nightly Builds:
  - EJB Application wird gebaut (EAR, eigene Integrations Tests)
  - Deployment des EARs auf einer Testmaschine
  - RCP Client Produkt wird gebaut, Build Parent enthält eclipse-test-plugin
  - Integrationstest startet Client Produkt
  - Umgeht den Login-Dialog (VM Arguments) und führt Tests aus
  - Reporting über Zustand des Builds im Jenkins (und E-Mail Trigger)

Schwierigkeiten.

Und Lösungen.

- Class Loading - Produkte - Releasing -

Class Loading Probleme

- Zwei Projekte (a und b), die EJB Remote Interfaces definieren
- Zwei Interfaces: X in Projekt a, Y in Projekt b
- Interface X extends Interface Y
- Einzige Abhängigkeit über Apache Maven zur Compilezeit (Dependency von a auf b)
- Problem mit dem jBoss EJBClient: Proxy-Generator generiert einen dynamischen Proxy aus X, zu dem zur Laufzeit nicht zwingend alle Imports von Y aufgelöst werden können

Equinox Class Loading

- Ein Bundle B versucht eine Klasse C in einem Package C zu laden:
  - P beginnt mit „java.“, return parent.loadClass(C)
  - P wird importiert, return exporter.loadClass(C)
  - P wird exportiert von einem RequiredBundle,
    - for each exporter
      - return exporter.loadClass(C) if found
  - C wird lokal gefunden, return C
  - C wird in einem Fragment gefunden, return C
  - P wird dynamisch importiert, return exporter.loadClass(C)
  - Buddy Loading ist aktiviert für dieses Bundle, return BuddyLoader.loadClass(C)
  - werfe eine ClassNotFoundException

Aus: McAffer, VanderLei, Archer: OSGi and Equinox, Addison-Wesley 2010

Buddy Class Loading

- Equinox kennt das Konzept des BuddyClassLoading
- Eclipse-BuddyPolicy: global
- Die so markierten Bundles können damit alle Klassen aus allen Packages laden, die global exportiert werden
- Nachteil: Untergräbt OSGi Modell, unter Umständen teuer (Performance)

Produkt

- „Bundle JRE for this environment with this product“
  - Lösung: JRE Feature beinhaltet JRE, wird über den Rootfile-Mechanismus während des Build ins Produkt kopiert
- Build Identifier in der Aboutbox
  - Lösung: maven-antrun-plugin ersetzt Platzhalter in bundle.properties der Application

Releasing

- Maven: maven-release-plugin
- Tycho: ?
  - tycho-versions-plugin, auszuführen auf Build Parent
    - Setzt Version in poms
    - Setzt Version MANIFEST.MFs, feature.xmls, .products...
  - Aber: Releases dürfen keine Qualifier haben (Patch?)
  - Vermisst: SVN Tagging (muss manuell getan werden)

Revolution?

Common Build Infrastructure

- Initiative innerhalb der Releng Gruppe
- Ziel: Make it easy to contribute!
  - Make it easy to build.
- Identifizierte Technologien (u.a.): Tycho, Maven, Hudson
- Eclipse Platform: Checkout, mvn clean install
- http://wiki.eclipse.org/Platform-releng/Platform_Build

Danke.