Turnaround cycle

Make a change

Check the change

Build, deploy, wait
Spring PetClinic Turnaround
TURNAROUND – WHY SHOULD YOU CARE?
Turnaround Cost

From over 15 projects and 150 people
• Average turnaround is at least 1 minute long
• Done about 5 times an hour

This sums up to
• 8.3% of total development time (1\*5/60)
• 3.5 hours a week
• Almost 1 work month a year
Programming is an exercise of the working (short-term) memory that holds the current context

Questions:

- How fast do you lose that context?
- How much time does context recovery take?
Working Memory

Interruption recovery time

[...] the recovery time after a phone call is at least 15 minutes.

– Interrupts: Just a Minute Never Is, IEEE Software, 1998

The time it takes the employees to recover from an email interrupt [...] was found to be on average 64 seconds.

– Case Study: Evaluating the Effect of Email Interruptions within the Workplace, EASE 2002

The recovery time for an instant message was estimated to be between 11 and 25 seconds

– Instant Messaging Implications in the Transition from a Private Consumer Activity to a Communication Tool for Business, Software Quality Management, 2004
1. With the recovery time considered, turnaround can easily cost more than 15% of total development time.
   
   • ~ 7 hours a week, 7 work weeks a year
   • This does not include the cost of quality degradation.
1. With the recovery time considered, turnaround can easily cost more than $15\%$ of total development time.
   - ~7 hours a week, 7 work weeks a year
   - This does not include the cost of quality degradation.

2. Every second counts! There is a significant difference between a minute, 30, 15, 5 and 1 second turnaround.
TRIMMING BUILDS
A typical web application build

1. Resolve dependencies
2. Copy static resources
3. Compile classes
4. Package modules in JARs
5. Package everything in a WAR/EAR
Exploded layout

• The project layout exactly follows the deployment layout

• All resources are edited in-place without copying
Automatic building

- Classes should be compiled automatically by the IDE
- The output should be set directly to WEB-INF/classes or similar
Deployment by linking

- The project is deployed by either pointing the container to it or creating a symbolic link in the deployment directory.

**Linux symbolic links**
- `ln -s`
- Symlinks can point to any file

**Windows symbolic links**
- Sysinternals junction utility on
A typical web application build

1. Resolve dependencies
2. Copy static resources
3. Compile classes
4. Package modules in JARs
5. Package everything in a WAR/EAR
Bootstrapping Builds

1. Can’t always use exploded layout

2. Instead:
   - Build the WAR/EAR
   - Unzip it to a temp directory
   - Remove some of the folders/jars and symlink them to the project folders
   - Set the project to build automatically

3. Easy to automate with a bootstrapping script

4. Save on copying resources and packaging classes
RELOADING CODE
Objects & Class Loaders ➔ Deployment, OSGi & etc ➔ JVM Dynamic languages
Reloading an Object

```java
OldClassLoader

MyObject.class

MyObject
```
Reloading an Object

OldClassLoader

MyObject.class

NewClassLoader

MyObject
Reloading an Object

OldClassLoader

MyObject.class

MyObject

NewClassLoader

MyObject.class
Reloading an Object

OldClassLoader

MyObject.class

MyObject

NewClassLoader

MyObject.class

MyObject

Recreate the object
Twin Class Loader

OldClassLoader

NewClassLoader

JVM
Twin ClassLoader Issues

New objects are not instances of old classes
- instanceof returns false
- Casting throws an exception

New classes are not members of the old packages
- Can get an IllegalAccessException when calling a perfectly legal method

Memory leaks are easy
- If you hold a reference to any object in the old classloader you will hold all old classes (including their static fields)
Web Deployment

Classes
Libraries
OldClassLoader
Servlet
App State
NewClassLoader
Web Deployment

- Session
- Classes
- Libraries
- Servlet
- App State
- New Classes
- New Libraries
- OldClassLoader
- NewClassLoader
Web Deployment

Session

Classes
Libraries

Servlet
App State

New
Classes
New Libraries

OldClassLoader
NewClassLoader

init()
Web Deployment

Session
- Classes
- Libraries

OldClassLoader

Servlet
- App State
- New Classes
- New Libraries

NewClassLoader

init()
Web Deployment

Serialize/deserialize

OldClassLoader

NewClassLoader

Classes

Libraries

Servlet

App State

Servlet

App State

New Classes

New Libraries

OldClassLoader

NewClassLoader
Web Deployment

- New ClassLoader
- New Classes
- New Libraries
- Servlet
- App State
- Session
- NewClassLoader
Web Deployment

Class loader scope
• Every deployed application gets a dedicated class loader

State recreation
• Application state is recovered by reinitialization
• Session state is (optionally) serialized and deserialized in the new class loader

Reloading time
• Application reinitialization time, typically around one minute

Problems
• Leaks memory
• Lazy caches need to be warmed up every time
Frameworks that implement the OSGi standard provide an environment for the modularization of applications into smaller bundles. [Wikipedia]
OSGi Redeployment

Classes
Libraries

Bundle
Module State

OldClassLoader
NewClassLoader
OSGi Redeployment

Classes
Libraries

Bundle
Module State

New Classes
New Libraries

OldClassLoader
NewClassLoader

www.devoxx.com
OSGi Redeployment

Classes
Libraries
Bundle
Module State
New ClassLoader
OldClassLoader
New Classes
New Libraries
Bundle

www.devoxx.com
OSGi Redeployment

Classes
Libraries

Bundle
Module State

New Classes
New Libraries

OldClassLoader

NewClassLoader

start()
OSGi Redeployment

Classes
Libraries

Bundle
Module State

New
Classes
New Libraries

Bundle
Module State

OldClassLoader

NewClassLoader

start()
OSGi Redeployment

- New Classes
- New Libraries
- Bundle Module State
- NewClassLoader
OSGi

Class loader scope
• Dedicated class loader per application module

State recreation
• Module state is recovered by reinitialization

Reloading time
• Module reinitialization time, usually less than whole application reinitialization

Problems
• Applications must be designed with OSGi in mind adding some overhead
• Rolling reloads – when a module is reloaded all dependent modules should also
Fine-grained Class Loaders

1. Wrap a class loader around components
   - E.g. Tapestry 5, RIFE

3. Very fast reloading
   - Few classes at a time
   - Components managed by the framework are usually easy to recreate
Component State

Class

Object

Old Component
ClassLoader
Component State

Old Component ClassLoader

Class

Object

New Component ClassLoader
Component State

Old Component ClassLoader

Class

Object

New Class

New Component ClassLoader
Component State

Class

Object

New Class

New Object

Old Component ClassLoader

New Component ClassLoader
New Component

Component State

New Class

New Object

New Component ClassLoader
Fine-grained Class Loaders

Class loader scope
- Class loader per component/service

State recreation
- State restored by framework (component/service recreated)

Reloading time
- (Almost) Instant

Problems
- Only managed components can be reloaded
- Managed components referring unmanaged code can be a problem (twin
Some Conclusions

1. Recreating the state is the breaking point of reloading a class
Some Conclusions

1. Recreating the state is the breaking point of reloading a class

2. Coarse-grained class loaders take too much time to recreate the state
Some Conclusions

1. Recreating the state is the breaking point of reloading a class

2. Coarse-grained class loaders take too much time to recreate the state

3. Fine-grained class loaders exhibit the twin class problem and are not universally applicable
Some Conclusions

1. Recreating the state is the breaking point of reloading a class

2. Coarse-grained class loaders take too much time to recreate the state

3. Fine-grained class loaders exhibit the twin class problem and are not universally applicable

4. Both are useful, but not really a solution to the zero turnaround problem
Dynamic Languages

- Class-based languages have same limitations as Java
  - Groovy
  - Jython
- Non-class based languages can have better support
  - JRuby
  - Clojure
HOTSWAP AND JAVAREBEL
HotSwap

OldClassLoader

MyObject.class

Code
101000101
100010010

MyObject

HotSwap

Debugger

www.devoxx.com
HotSwap

User saves class from IDE

OldClassLoader

MyObject.class

Code
101000101
100010010

MyObject

HotSwap

New Code
111000100
101010010

Debugger
HotSwap

OldClassLoader

MyObject.class

New Code
111000100
101010010

HotSwap

MyObject

Debugger
HotSwap

Updates classes and objects
• Almost instantly
• Can be attached remotely

Very limited
• Only updates method bodies, no new fields, methods or classes
• Needs a debugger session running,
JavaRebel Approach

Classes
Libraries
Objects and Code

ClassLoader
ClassLoader
ClassLoader

Reloading “Interpreter”

JVM
JavaRebel Agent
OldClassLoader

MyObject.class
  Code
  101000101
  100010010

MyObject

New Code
  111000100
  101010010

JavaRebel agent

MyObject.class file changed

JavaRebel
JavaRebel

OldClassLoader

MyObject.class

New Code
111000100
101010010

MyObject

JavaRebel agent

www.devoxx.com
JavaRebel Installation

1. -noverify -javaagent:/path/to/javarebel.jar
   • Enables the JavaRebel agent
   • All **.class** files in the classpath will be monitored for changes automatically

3. (Optional) -Drebel.dirs=folder1,folder2,…
   • Specifies IDE output folders or just class folders
   • Can deploy a WAR/EAR and still get instant updates to code
DEMO:
Petclinic With JavaRebel
JavaRebel

Just works
• No configuration necessary!
• Runs on all JVMs starting with 1.4
• Supports all major containers

Seamlessly
• Changes are visible in reflection
• Serialization works as usual
• Dynamic proxies work as usual
JavaRebel

- Commercial tool, free 30 day trial
- No free/open source analogs
- Get it from: www.zeroturnaround.com
  or just google “javarebel”

- Personal license:
  ~ $10

- Commercial license:
JavaRebel History

1. JavaRebel 1.0 released in December, 2007

2. Over 10 000 licensed users

3. Some of our customers:
   - LinkedIn
   - Turner
   - Roche
   - Logica
   - Disney.com
AND BEYOND
JavaRebel

MyObject.class file changed

OldClassLoader

MyObject.class

Code
101000101
100010010

JavaRebel agent

New Code
111000100
101010010

Configuration (XML, annotations, …)

MyObject
JavaRebel

*OldClassLoader*

*MyObject.class*

**New Code**

```
111000100
101010010
```

*Framework*

*MyObject*

*Configuration (XML, annotations, ...)*

*JavaRebel agent*
Types of Configuration

Service Glue
- EJB 2.0/3.0
- Spring
- Guice

Web Controller
- Struts 1.0/2.0
- Stripes
- Spring MVC

ORM
- Hibernate
- TopLink
- JPA
JavaRebel Plugins

Open Source JavaRebel SDK
- Plugins are found and started from classpath
- Javassist support allows patching

Spring Plugin
- Adding/removing beans dependencies via setters/fields
- Adding new beans via XML or
DEMO:
Petclinic With JavaRebel Spring plugin
More JavaRebel

More plugins
• **Available:** Spring, Guice, Struts 2, Tapestry 4
• **Coming:** Stripes, Wicket, Struts 1,

Virtual Classpath
• All the benefits of exploded development with unexploded one
• Automatically maps propagates
Take Away

1. Every next second spent on turnaround costs more!
1. Every next second spent on turnaround costs more!

2. **Builds** should be as slim as possible, **symlink** is your best friend
Take Away

1. Every next second spent on turnaround costs more!

2. **Builds** should be as slim as possible, **symlink** is your best friend

3. **Code reloading** is a complicated problem with HotSwap, OSGi and framework support being the best partial solutions available for **free**
Take Away

1. Every next second spent on turnaround costs more!

2. **Builds** should be as slim as possible, **symlink** is your best friend

3. **Code reloading** is a complicated problem with HotSwap, OSGi and framework support being the best partial solutions available for **free**

4. **JavaRebel** solves most of turnaround problems for a cost
But what about production?

1. A twin problem of Zero Turnaround is **Zero Downtime**
   - How to update the application without stopping it even for a second?

2. **JavaRebel** provides an engine that can reload changes to classes instantly

3. **LiveRebel Server** provides a management server and console to put it all together
For your chatting pleasure:

1. Join the Proximus Wireless LAN

2. Browse to http://81.169.3.48:3000/chat

3. Chat with other attendees and be sure to hold the browser open during the end of the session
   • Try the /help command
LiveRebel Server
Thanks for your attention!

http://www.zeroturnaround.com/javarebel/

For LiveRebel Server private beta: http://www.zeroturnaround.com/beta/