Clone Clone Make: A better way to build

Dan Heidinga, J9 VM Interpreter Lead
Daniel_Heidinga@ca.ibm.com
@DanHeidinga
16 November 2016
Important disclaimers

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

- WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

- ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

- ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

- IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

- IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
  - CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS
Who am I?

- I've been involved with virtual machine development at IBM since 2007 and am now the J9 Virtual Machine Team Lead. J9 is IBM's independent implementation of the JVM.

- I've represented IBM on both the JSR 292 ('invokedynamic') and JSR 335 ('lambda') expert groups and lead J9's implementation of both JSRs.

- I've also maintain the bytecode verifier and deal with various other parts of the runtime.
Overall question

How do you transform from the organization you have to the one you want?
– Conway’s law in action
– Understand the past allows you to know what and how to change
A bit (more) of my background

- I’ve worked on the J9 team for close to 10 years now
- In that time, we’ve gone through a number of build system migrations, some successfully, some not so much
- Worked with many different source control systems from Envy/Smalltalk, CVS, RTC, & now GIT
- This talk will look at the lessons we’ve learned about the importance of your build pipeline and why it’s essential for transforming your organization.

- I’m going to caricature some of these issues so apologies to my team for any inaccuracies / oversimplifications.
Conway’s law

organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

— M. Conway

https://en.wikipedia.org/wiki/Conway%27s_law
Conway’s law in action

Bengaluru

Hursley

Toronto

Ottawa
Conway’s law in action

Bengaluru

Class libraries

Hursley

Project management

JIT

Toronto

Ottawa

VM / GC
Delivery pipeline

JIT → VM → GC → Composition → SDK

Class libraries
Delivery pipeline: reality
<table>
<thead>
<tr>
<th>Architecture</th>
<th>Operating System</th>
<th>Bit width</th>
</tr>
</thead>
<tbody>
<tr>
<td>X86</td>
<td>Windows</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
<tr>
<td>Linux</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
<tr>
<td>PPC BE</td>
<td>Linux</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
<tr>
<td>AIX</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
<tr>
<td>S390</td>
<td>zOS</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
<tr>
<td>zLinux</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 + CompressedRefs</td>
</tr>
</tbody>
</table>
Large milkweed bug (Oncopeltus fasciatus) adult and two juveniles. Photo by Greg Hume. https://commons.wikimedia.org/wiki/File:Oncopeltusfasciatus.jpg
Minimum bar to submit code

Build silos

Integration build

VM → c build → p build

GC → c build → p build

JIT → c build → p build

SDK → c build → p build
Delivery pipeline updated

Class libraries

Integration build

Composition

JIT

VM

GC

JIT

SDK

SDK
Lesson: Foster a “1 team” approach

Teams should be invested in the product’s success, not just components.

Local solutions (walls) hurt global cooperation.

Customers care about the product, not components.
Mounting problems

- Cross team squabbles
- Slow promotion rates
- Trouble making cross component dependent changes quickly
- Complexity

- We should have been more scared of this.
The Joel Test

1. Do you use source control?
2. Can you make a build in one step?

http://www.joelonsoftware.com/articles/fog0000000043.html
Enter the delayed release

- You discover the product about to ship is broken: How long to turn around a fix?

- Our complex build system fails the Joel Test
  - Long turn around cycles
  - Many steps
Development Transformation

- Enter feature teams
  - Each major feature gets its own branch of each component
  - Minor bug fixes are their own feature branch

- Enter release trains
  - Feature streams merge into a validation stream periodically
  - Validation stream closes to run SVT
Dev Transformation: Feature streams + release train
Dev Transformation: Feature streams + release train

Different components, different models
High communication overhead
Long period between merges

SVT
Lessons from failure

If teams are not fully committed to the same model, it’s doomed to fail.

The developer’s using the system need to be the ones to define it.

Complexity kills. In both software design and build systems, the complex one will fail first.

Optimize for the right thing: speed of innovation.
Pace of innovation in our industry is increasing

- Technology landscape is changing.
- Whoever innovates faster will win
- Innovation requires simplicity.
  - Aggressively remove complexity from system, processes, products
Recap

- **Quality bars and local silos**
  - Teams should be invested in the product’s success, not just their components.
  - Local solutions (walls) hurt global cooperation.
  - Customers care about the product, not components.

- **Feature streams and release trains**
  - If teams are not fully bought the same process, it’s doomed to fail.
  - The developer’s using the system need to be the ones to define it.
  - Complexity kills. In both software design and build systems, the complex one will fail faster.
  - Optimize for the right thing: speed of innovation.
clone clone clone make
Simple motto increases alignment

- Motto provides focus to the team and directs decision making
  - Agile refrain: highly aligned, loosely coupled teams
  - Goal needs to be important to both the business and developers

- Agile succeeds when the goal your team cares about is front and center
  - If the goal is "agile transformation“, you lose

- This needs to be something everyone on the team understands

- Easy to guide decision making when you can tie it back to the vision
External dependencies

JIT → VM → GC → Composition

OpenJDK → Class libraries → Composition

Composition → SDK
Patches on upstream are technical debt

- If you patch / modify a component, get that team (internal or external) to accept your changes as fast as possible.

- If the upstream changes, your patches need to be modified to track it

- Every patch is technical debt, even the ones that add value!
  - If you have a “valuable” patch, exploit for a time and then upstream it
  - Eventually someone will add equivalent functionality. Easier if you just get yours accepted.
External dependencies

OMR

JIT

VM

GC

Composition

 SDK

OpenJDK

Class libraries
Eclipse OMR Mission

Build an open reusable language runtime foundation for cloud platforms

– To accelerate advancement and innovation

– In full cooperation with existing language communities

– Engaging a diverse community of people interested in language runtimes
  • Professional developers
  • Researchers
  • Students
  • Hobbyists
Eclipse OMR
Created March 2016

http://www.eclipse.org/omr
https://github.com/eclipse/omr

Dual License:
Eclipse Public License V1.0
Apache 2.0

Users and contributors very welcome
https://github.com/eclipse/omr/blob/master/CONTRIBUTING.md
Working in the open

- Refactoring Eclipse OMR out of J9 required a massive change to how we work
  - Working in the open. No hiding behind component boundaries.
  - External communication on issues / mailing lists over internal communication
    - Easier to onboard remote team members: anyone can modify anything

- Pull request contribution model
  - Anyone can open a pull request to any of the code in OMR
  - If it passes the tests and the committers review, it gets merged

- Using the same processes internally as externally.
  - Already driving changes by increasing the openness of our teams
  - Doesn’t matter which geography you’re in, or even if a member of our team
Take the Best from Open Source

> git checkout -b <feature branch>
> make all
> make test \textbf{(Initial quality bar)}
> git add <files>
> git commit -m "commit message"
> git push

- Pull request opened from my fork to upstream
- Continuous integration build validates merged pull request \textbf{(quality bar)}
- Committers review & merge if build passes
Source changes & internal tools

- Look at your tool chain
  - How many dependencies / required tools can be installed easily?
  - Are they available to everyone?
  - How many different ways of doing essentially the same thing?
    - make, ant, perl, python, shell scripts, configure, makefile generators

- Example: Rewrite the core of the interpreter from generated ASM to C++
  - High barrier to change:
    - custom tools
    - only buildable on Windows
  - Even if it comes at a cost
Co-dependent components aren’t components

OMR → VM
      → GC
      → JIT

OpenJDK → Class libraries

Composition → SDK
Elephants are too big to eat at once

- Trying to replace everything at once is too much
  - Pick one platform and make the process work end to end there
  - Simplicity is key again: fewer dependencies makes this easier
    - But pick the right ones…. Those that are supported across all required platforms

- Different MVPs are possible
  - OS / architecture / distro combos

- Incremental steps: how fast can you turn the crank?
Start using the new system as soon as possible

Get the new system in people’s hands, even if it's not 100% functional
– building the new system while developing the old is hard
– the sooner you make the switch, the better

- Know the difference between simple and simplistic
  – Do you need to mix in binaries from other sources?
  – Other internal requirements?
“But I’ve always done it this way”

- Social as much as technical change

- Step by step to get to the new system
  - Will be complaints along the way
  - But better than replacing the entire world at once (elephant)

- Be aware of potential cost to morale
Clone, Clone, Make for Java in Action

> git clone <ibm gitlab repo>
> cd openjdk
> sh ./get_source.sh --with-j9
> sh configure --with-boot-jdk=<...> --with-j9
> make all
> ./java -showversion HelloWorld

openjdk version "9-internal"

OpenJDK Runtime Environment (build 9-internal..jdk9)
IBM J9 VM (build 2.9, JRE 1.9.0 Linux amd64-64 Compressed References)
J9VM - R29_20160528_0102_B305707 (Pure C)
JIT - tr.open_20160527_118242_8ba92ef8.green
OMR - d47907ca)
Hello World
Clone clone make drives simplification

Clone clone make vision drives simplification into the

- Build pipeline
- Organization structure
- Breaks down component walls
Our end goal: OMR + OpenJ9

Eclipse OMR

Communities Beyond Java: Eclipse OMR

Open JDK

IBM SDK for Java

Proven adaptable technology in the open for rapid innovation and collaboration across multiple language communities

Invent Your Own Language!

Java community open innovation and collaboration, deep platform exploitation for X86 & IBM hardware platforms (OpenPOWER, Linux ONE)

Long term support, quick response for problems, and other forms of IBM customer specific engagement

Invent Your Own Language!
IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.

Java and all Java-based marks, among others, are trademarks or registered trademarks of Oracle in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

THE INFORMATION DISCUSSED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, SUCH INFORMATION. ANY INFORMATION CONCERNING IBM'S PRODUCT PLANS OR STRATEGY IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.
IBM is open sourcing its J9 JVM technology
  - Includes Testarossa Just in Time (JIT) compiler

Eclipse OMR project is leading edge: github.com/eclipse/omr
  - Project created March 7, 2016: ~300KLOC
  - Compiler contributed September 16, 2016: ~500KLOC

Open J9 project is also coming
  - We’re working on it at same time as Java 9 development