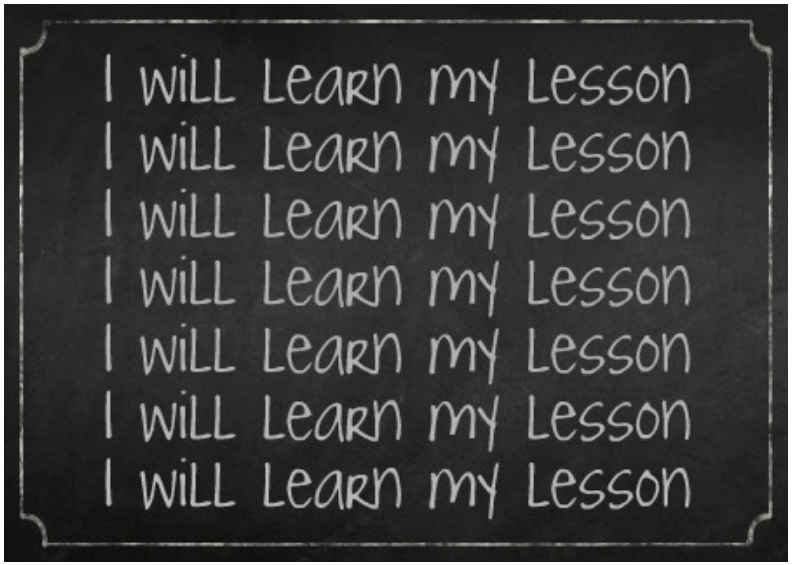
Lessons Learned in Software Testing

QA infrastructure –
maintaining robustness in OS-like commercial software



Marcus Lagergren

@lagergren

marcus[at]lagergren.net

About me

- M. Sc. from KTH, Stockholm
- I know a lot about runtimes and operating systems. Talk to me about low level stuff.
- I love power tools, heavy metal and scuba diving

Agenda

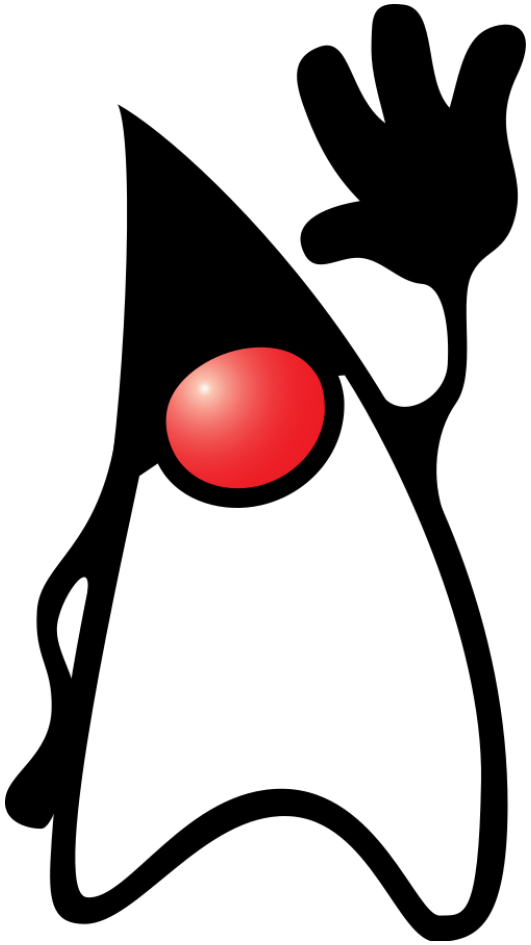
- Robustness in special/commercial apps
- Build systems
- Source control and Development
- Tests
  - Functionality, Performance, Regressions

Agenda

- Result databases
- Automatic testing
- Complex and not-so-standard testing
- Development
- Battle scars

Why listen to me?

10 years in JVM development



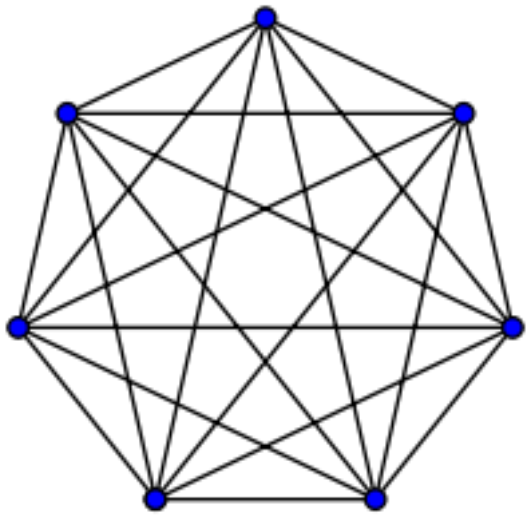
Why listen to me?

4 years in Operating Systems development

Virtualization/Hypervisors

Runtimes are hard

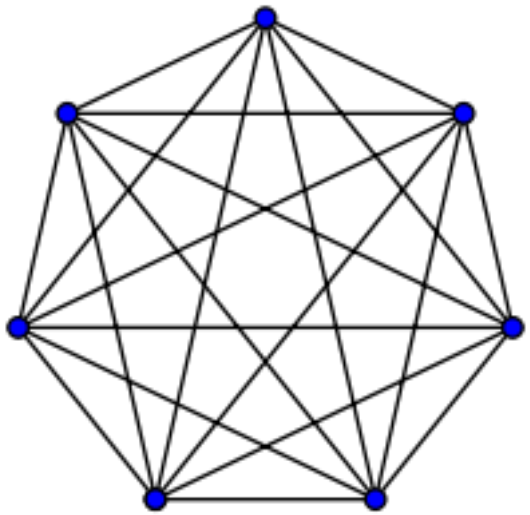
Harder to debug software hardly exists



Runtimes are hard

Harder to debug software hardly exists

Tight coupling

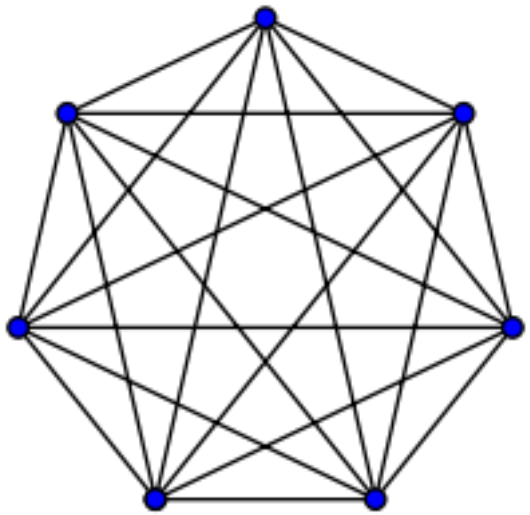


Runtimes are hard

Harder to debug software hardly exists

Tight coupling

Operating System like behavior



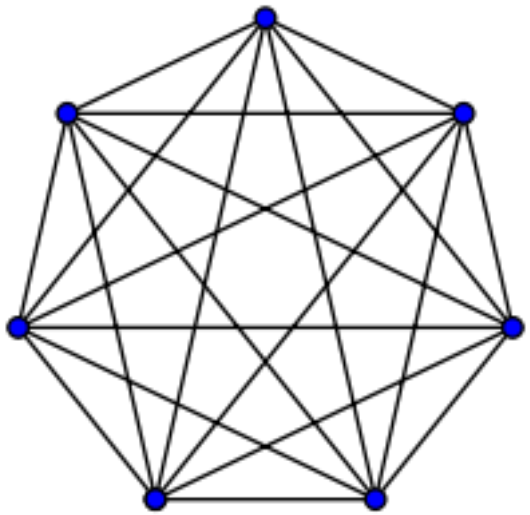
Runtimes are hard

Harder to debug software hardly exists

Tight coupling

Operating System like behavior

Extreme test demands



Building a JVM in 1998

“This is the second year in a row that Sun has had the same HotSpot presentation at JavaOne. Our customers can’t really wait. Let’s roll our own – how hard can it be?”

Building a JVM in 1998

hubris

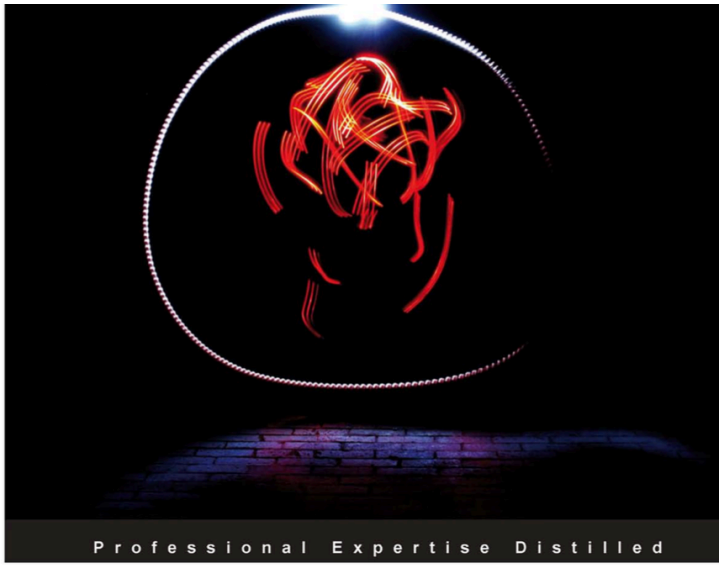
/ˈhjuːbrɪs/

noun

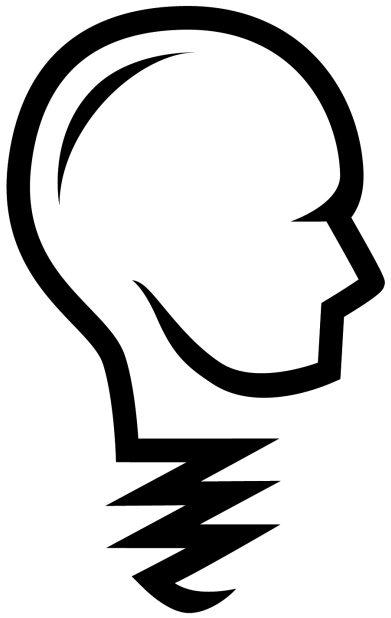
excessive pride or self-confidence.
"the self-assured hubris among economists was shaken in the late 1980s"
synonyms: arrogance, conceit, conceitedness, haughtiness, pride, vanity, self-importance, self-conceit, pomposity, superciliousness, feeling of superiority; More

- (in Greek tragedy) excessive pride towards or defiance of the gods, leading to nemesis.

Building a JVM in 1998

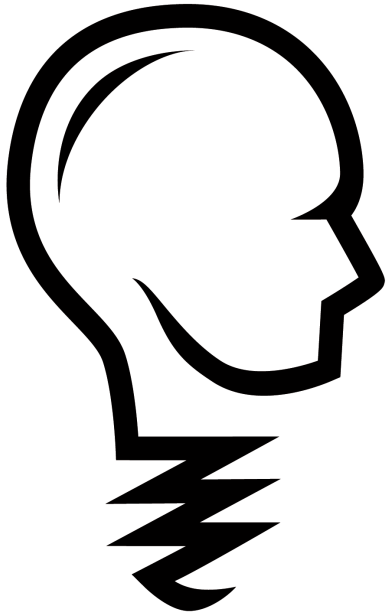


A strange beast; We've had to invent stuff from day 1



A strange beast; We've had to invent stuff from day 1

(ok from day 365 or so, but lessons learned)



We’ve made many mistakes along the way

We've made many mistakes along the way

We had to invent new ways to test tightly coupled systems

We know what “QA Utopia” looks like

(But maybe not how to get there every time)

QA Infrastructure

- QA infrastructure is harder than development infrastructure
  - It is also more important

QA Infrastructure

- QA infrastructure is harder than development infrastructure
  - It is also more important
- Valuable lesson: *DEVELOP IT PARALLEL TO THE APPLICATION*

QA Infrastructure

- QA infrastructure is harder than development infrastructure
  - It is also more important
- Valuable lesson: *DEVELOP IT PARALLEL TO THE APPLICATION*
  - Not necessarily TDD
  - Be a little bit wary of “rituals”

QA Infrastructure

Sometimes the boundaries between app and test infrastructure aren’t even clear

QA Infrastructure

- QA and Dev should NOT be separate departments or roles
- They should always work together
  - Preferably physically close
- They should be able to do each others' work.

QA Infrastructure

VERY DANGEROUS to have a separate QA department on another floor or another continent

QA Infrastructure

VERY DANGEROUS for QA to just do blackbox testing without understanding what's in the box

QA Infrastructure

Real world example: running hundreds of JVM tests with different garbage collection flags. But they never garbage collect!

QA Infrastructure

In my team, there is no difference between a developer and a QA engineer.

Do not treat them differently!

Build System Basics

- Build system, test system, source control
  - Should be parts of the same distributed system

Build System Basics

- Build system, test system, source control
  - Should be parts of the same distributed system
- Mobility
  - Build anything anywhere, locally or globally (distributed) – “A distributed cross compiler”

Build System Basics

- Build system, test system, source control
  - Should be parts of the same distributed system
- Mobility
  - Build anything anywhere, locally or globally (distributed) – "A distributed cross compiler"
- Self contained and part of source control
  - Build kits
  - Bit identical builds in virgin laptops
  - Just type "`make`"

Test System

- Also under source control
  - We like tests and code in same repo
  - This is the norm these days
- Distribute, virtualize, maximize resources
- Local and remote test runs
  - Identical results on identical hardware

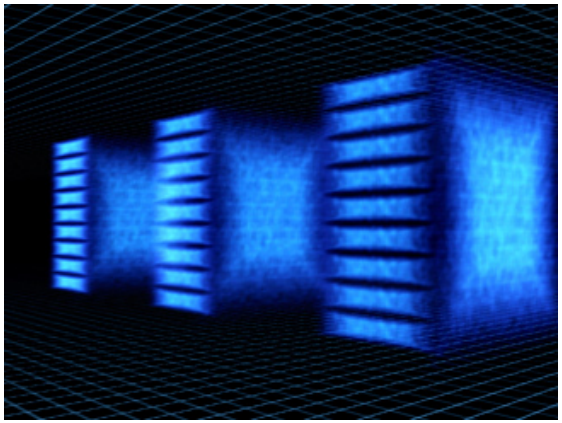
Test System

- Test machines
  - Dedicated performance test machines
  - Virtualized functionality test machines
  - Any machine can volunteer cycles for testing
  - Easy to add new machines

Test System

Virtualization is lovely!

Spin off any complex test environment from images. Crunch. Throw away.



Building Blocks - Tests

- Regression tests
  - Main method and return value
  - Keep it simple
  - “Spent a few hours distilling this huge program down to a reproducer of BUG 123456”

Building Blocks - Tests

CLAIM: If it's simple to write and submit a test, ~> 50% of the bugs can get regression tests added as part of the original bugfix.

(I will address the other 50% later;

now we talk about well behaved deterministic tests)

Result Database

- Put results in a cheap database with sensible layout
  - Any freeware is fine – get it up and running!
- Easy to maintain and backup

Result Database

- Query from local machines about historical test results
  - “When exactly did this performance regression appear?”
  - “List all benchmark scores on this machine for this benchmark since January 1”
  - “Has this functional test failed before? What were the bugfixes?”

Source code

- Cross referencing source code from issues is awesome
  - Crucible
  - Custom solutions
- Distributed code reviews
  - Not just mailing lists

Performance Testing

- Anything can affect performance
- EVERYTHING affects performance
- We need automatic regression warnings
  - Anyone who submits a performance regression gets an e-mail from the test system
- *Make it easy to continuously add more benchmarks*
- Automation: deviations, baselines, invariants

Performance Testing

- Continuous automatic testing
- For example:
  - The product for Solaris has been made available off and on over the years
  - Bit rot sets in immediately when removed from automatic testing

Testing

- Release version may break debug version
  - And vice versa
- Linux version may break Windows version
  - And vice versa

```
./bin/localtest.sh
./bin/remotetest.sh
```

Testing

Use *ULTRA-PEDANTIC* compiler flags!!!

Warnings are errors! *Pedantic*! Enforce whatever safety conventions you have agreed upon. Don't wait

Testing

If it fails, what's the command line reproducer?

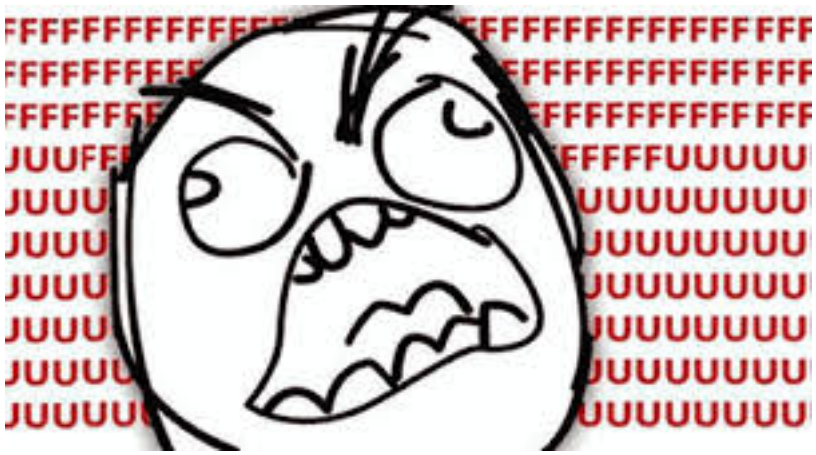
Testing

If it fails, what's the command line reproducer?

No seriously – give me the command line reproducer. No excuses!

Testing

No – I don't want that huge log with `maven` running something in the background and 250 environment variables



What about the other 50%?

What about the other 50%?

Simple programs with “`main`” functions are not be enough for all reproducers

(of course)

What about the other 50%?

- How do we test a specific optimization in a compiler?
  - It only fails on huge IRs after hours in production

What about the other 50%?

- How do we test a specific optimization in a compiler?
  - It only fails on huge IRs after hours in production
- How do we test a boundary condition on the heap that crashes the GC?
  - It only crashes after hours in production

What about the other 50%?

Key observation:

We need to export a *state*!

What about the other 50%?

Key observation:

We need to export a *state*!

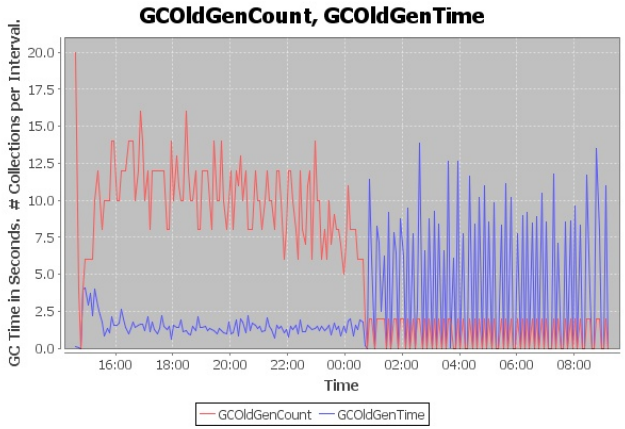
Whitebox testing

Examples

Create a heap state with a few objects in nasty places.

Inject it into the runtime.

Trigger a garbage collection.



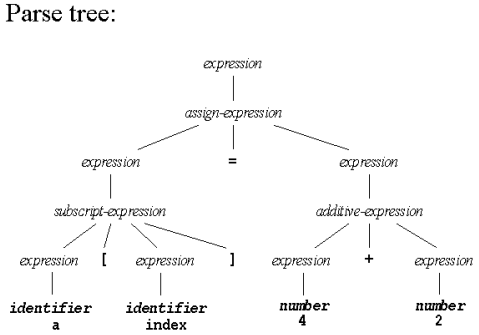
Examples

Serialize an IR from just before an offending optimization.

Inject it into the runtime.

Trigger the optimization.

Save the resulting IR and compare to reference.



State

Modularize enough to enable whitebox testing

But of course it’s never as simple as that

Parallelism

Multithreaded apps?

Race conditions?

Parallelism

Multithreaded apps?

Race conditions?

How do we even make test cases?

Parallelism

Synchronization points.

Randomized input.

Randomized sleeps.

Try to cover the malicious side effects of parallelism.

Static analysis (e.g. RaceTrack algorithm)

Parallelism

We are rapidly moving from explicit to implicit parallelism.

Great for the programmer.

Still hard for the runtime.

A Pinch of Terror Testing

- AllocAndRun
- RedefineClasses
- GoToSupervisorMode
- ExceptionInClinit
- Deoptimize a lot
- “Kitchensink”

Crash Once - Brute Force

Some times you just need to crunch for a long, long time.

Nothing else suffices to reproduce a problem.

(or the framework for whitebox testing can't possibly exist. Heisenbugs)

Crash Once - Brute Force

Use all free CPU cycles in your machine park.
Crunch!

Make dumps full and comprehensible.

Crash Once - Brute Force

DISK IS CHEAP!

SAVE EVERYTHING!

ALWAYS!

Crash Once - Brute Force

"Phone home". Inflict it on your own org during rampdown when dogfooding.

The Platform Matrix

Maximize amount of common code!

Choice between platform specific features and test matrix growth

The Platform Matrix

Maximize amount of common code!

Choice between platform specific features and test matrix growth

Example: *“platform independent native code stubs”*

Example: *“two vs one tiered JIT compiler”*

The Platform Matrix

Beware of “false abstraction”

The Platform Matrix

Beware of “false abstraction”

“That extra parameter is always `null` on Solaris”

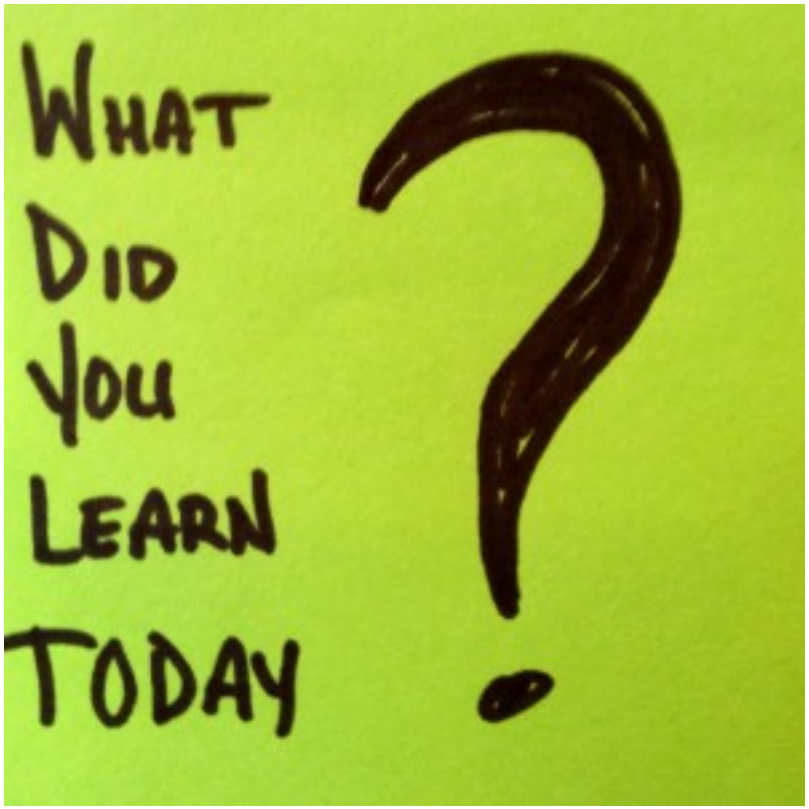
Development & Policy

Don’t be too much of a quality fascist when code is written

Development & Policy

The infrastructure should *quickly* and *mercilessly* raise the alarm as something breaks, to prevent further damage

Prevent promotion of broken bits


WHAT
DID
YOU
LEARN
TODAY
?

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”
- There is really no boundary between QA and Dev work. Keep the team together.

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”
- There is really no boundary between QA and Dev work. Keep the team together.
- Think about whitebox testing all the time

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”
- There is really no boundary between QA and Dev work. Keep the team together.
- Think about whitebox testing all the time
- If the system requires tight coupling, don’t be afraid to do that. But think about how to test.

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”
- There is really no boundary between QA and Dev work. Keep the team together.
- Think about whitebox testing all the time
- If the system requires tight coupling, don’t be afraid to do that. But think about how to test.
- Make it easy to add and write tests

Lessons Learned - Summary

- Build the test infrastructure in parallel with the application
- TDD might not be the whole answer. Avoid “religions”
- There is really no boundary between QA and Dev work. Keep the team together.
- Think about whitebox testing all the time
- If the system requires tight coupling, don’t be afraid to do that. But think about how to test.
- Make it easy to add and write tests
- Use all available CPU cycles for testing

Acquiring Sun

Finally – A little horror story

Acquiring Sun

Finally – A little horror story

With a happy ending

[Sequels still to follow ;-)]

Acquiring Sun

Acquiring Sun

Acquiring Sun

[A small horror story from 2010-2015]

Preface: ***[JRockit just became nicely modular, Sun betting the company on JavaFX, No new Java since 2006, fire all the QA, No code standard, afraid of change, cheap QA in a different country, no build kits, No Windows development, no STL because Solaris compiler, No IDEs, "IncludeDB", Least common denominator everywhere, Tests that never garbage collect run with 10 different GC flags, Huge checkin overhead, Open source code reviews "I don't like your variable name", but never "I applied your patch, ran some tests, found a deadlock" – NEVER!]***

Fix all this ☺

Acquiring Sun

[A small horror story from 2010-2015]

Today: `./configure; ./make images`

Major test cleanup

IDE support for all platforms

OpenJDK build takes 5 minutes, not 45.

Whitebox testing introduced e.g. for GC

It's always

orders of magnitude harder

and

orders of magnitude more time consuming

to retrofit good testing than to develop it continuously. Know and feel what QA means.

Q & A

@lagergren