Dismantling Technical Debt and Hubris

@ShelleyMLambert  JPoint 2021
The Method behind the Madness

- Definitions
- Various (often conflicting) requests
- 4 modes of activities
  - Maintenance
  - Development Transformation
  - Innovation
  - Open-source Readiness
- Measuring success
Definitions

Technical Debt

- cost of additional future work caused by implementing an easy (limited) solution instead of using a better approach that would take longer

Hubris

- excessive confidence or arrogance

- leads to lack of questioning and planning, short-sighted, harmful behaviour
The Noise

- Reduce build time
- Test smarter
- Test faster
- Support continuous delivery
- Support other teams
- Ensure tests can run anywhere
- Make testing easier
- Make triaging features faster
- Simplify processes related to testing
- Prepare some tests for open-sourcing (1st class citizen)
- Keep defect escapes low despite reducing amount of testing

We are not doctors, but we play them on TV at work
Context and Stats

- Over quarter billion downloads
- Rapidly growing install base
- ~93,000,000 tests run per release, massive-scale devops
- Growth & Transformation while Maintaining Pace

Market share (JDK8, March 2020)
blog.newrelic.com/technology/state-of-java

dash.adoptopenjdk.net
Making Sense of the Noise

- Legacy Tests for Java
  - Maintenance
- New Tests
  - Development Transformation
- New Ways of Testing
  - Innovation
- Open & Transparent
  - Open source Readiness

- Assess and cull
- Fix test defects
- Maintain and improve tests
- Simplify test processes

- Reduce defect escape rate
- Change-based testing
- Keep up with new features

- CTD
- Defect Injection
- Deep Learning

- Minimal tests & tools
- Functional coverage
- Improved workflows
- Portable
Legacy Tests

functional
- testNG, cmdlinetester

openjdk
- Jtreg, testNG

system
- STF
- junit, sbt & others

external
- perf
- Assorted benchmarks

jck
- javatest
The Problem with too many frameworks

- Increased automation complexity
  - Auto-exclude/re-include
  - Reporting
  - Auto-triage
- Onboarding learning curve
- Documentation
Consolidate

- functional
  - testNG, cmdlinetester

- openjdk
  - Jtreg, testNG

- system
  - STF

- external
  - junit & others

- perf
  - Assorted benchmarks

- jck
  - javatest

TestKitGen

make _functional
make _openjdk
make _system
make _external
make _perf
make _jck
3-layer Cake of Automation

**Test Results Summary Service (TRSS)**

Responsibilities:
- Categorize logically, Generates test targets based on playlist (level/platform/version/impl specific)
- Execute tests via common command line / make target patterns
- Test addition via auto-discovered directories, Standardize exclusion
- Parameters: BUILD_LIST, EXTRA_OPTIONS, JVM_OPTIONS, TEST_FLAG, ITERATION
- Generate dynamic test playlists based on input (smart parallelization)

**CI System (Jenkins / Tekton / AzDO / Github Actions, etc.)**

Responsibilities:
- Schedule regular builds
- Multiplex tests across multiple nodes of all platforms
- Basic GUI view of test results
- Basic forms of parallelization

**testkitgen (TKG)**

Responsibilities:
- Monitor multiple CI servers
- Graphical, Aggregate summary, Deep history
- Search/sort/filter
- Pluggable parsers
- Basis for deeper analytics and deep learning services

Layer 3: Requires L1 & L2
- Database queries
- Basis for extras, search /sort / filter /analyze across DB entries
- Monitor anything

Layer 2: Requires L1
- More nodes
- Scheduling
- GUI
- enhanced reporting for tests that support Junit output

Layer 1: Standalone
- Execution
- Exclusion
- Test target report summary (TAP)
- Reproducibility
Maintenance Mode

“Stuff was always broken. The code was broken, the tests were broken, and the team was broken! I knew what it meant to be buried under quality and technical debt where every innovative idea was squashed lest it risk breaking the fragile product it depended on.” *

- **Demotivating** work to maintain legacy tests and ailing infrastructure

- Try to **minimize effort** in this mode by:
  - assessment
  - removal
  - fixing defects
    - estimated ~ 1 out of 10 test defects hide a real defect
    - if too difficult to fix/maintain, replace/remove

* From “How Google Tests Software”
Development Transformation Mode

“The secret of change is to focus all of your energy not on fighting the old, but on building the new.” *

* Socrates

Test and test framework reduction:

- Reduce before tests are written - Combinatorial Test Design
- Reduce number of frameworks
- Reduce after from existing pool of tests - Change-based testing based on code coverage information on PR builds

Improve work flow:

- Remove ‘friction’, less time per change, less error prone
- Address long-standing legacy problems - tools that created more work (problem tracking)
Innovation Mode

“Innovation is creativity with a job to do.” *

* John Emmerling

- testkitgen (TKG) - thin veneer to make the large variety of tests to behave the same
- Continuous refinement - building blocks for future goodness
- Test Results Summary Service (TRSS) - visualizations and features to cope with massive amount of test data
Open Source Readiness Mode

“Creating community involvement around a QA project is one of the best things you can do for your product. It puts the product before interested users early on in the process, and allows them to use and experiment with in-development features ... creating a tribe of people who will follow your product. That tribe will do far more evangelism and word-of-mouth promotion than any glossy advertisement could hope to accomplish.”

* From “Building Open Source QA Communities of Beautiful Testing”

- Use open-source tools (avoid proprietary, in-house solutions)
- Minimal, simple and fast test suites, assure functional coverage (CTD), continuous delivery schedule
- Clean, public-ready test code, 1st class citizen of same quality as product code, test quality, tools & process attract contributors
How Can We Measure Success?

<table>
<thead>
<tr>
<th>Mode</th>
<th>Success Metrics</th>
</tr>
</thead>
</table>
| Maintenance             | • size of test defect queue  
                          | • # of frameworks to manage  
                          | • # of steps for common tasks |
| Development Transformation | • execution time of test jobs  
                          | • # of tests in test groups  
                          | • # of defect escapes  
                          | • speed/ease of adding tests |
| Innovation              | • # of intermittent defects  
                          | • average defect triage time |
| Open Source Readiness   | • # of contributors to the open source project  
                          | • speed/ease of adding tests  
                          | • # of defects |

**Audience Participation**

For each suggested metric, what direction is better?

Which is best if bigger or smaller? # of steps for common tasks, execution time, # intermittent defects, # of contributors
Maintenance Mode Success Metrics

- Size of defect queue, backlog burndown
- Number of steps required to do common tasks
- Number of frameworks to manage (example, 20 to 5)
20 Frameworks to 5

No time for war stories

Ask me later!
Development Transformation Success Metrics

- Execution time of test jobs
- # of tests before & after change-based testing
- # of defect escapes (example, defect escape analysis)
- Speed & ease to add/execute tests (example, TKG refinements)
Breakdown of Problem Reports - Defect Escape Analysis

By Problem Type:
- Crash (17)
- Functional Issue (18)
- Unexpected behaviour (11)
- Hang (9)
- Security (5)
- OOM (2)
- Other (17)
- Performance (4)
- Usability (3)
- Documentation (2)
- Standards non-compliance (3)
- Exception (3)

When Found:
- Development (40)
- Component Test (1)
- SVT (25)
- Perf (2)
- Deployment (12)
- Webs_deployment (7)
- Other (7)

First chance:
- Code review (7)
- Component Test (6)
- Development (27)
- Customer (3)
- JCK (1)
- Perf (2)
- Other (3)
- Not specified (45)
TKG example - faster, better, smarter

- **Flexible** play and granularity
  slice’n’dice via playlists tags

- **Standardized** output
  enforced TAP output

- **Reproduce/rerun**
  failed.mk, rerun in Grinder, SHA.txt

- **Automation**
  Auto-detect & MachineInfo
  Auto-disable/re-enable
  PARALLEL=Dynamic|BySubdir|Iterations|None
Innovation Success Metrics

- Track specific goals of the prototypes
  - Deepsmith: reduce # of intermittent defects
  - Bug Prediction & Core Analytics: predict crashes/bugs
  - Deep AQAtik: reduce average defect triage time (example, deep AQAtik)
Deep AQAtik

- Utilize deep learning model to recommend possible issues related to test failures

- Github, JBS, StackOverFlow, Support issues

**Flowchart:**

1. Collect Issues from GitHub repos
2. Pre-process issue contents
3. Train DL model to classify multiple issues
4. Evaluate DL model with TRSS/Jenkins output
5. Deploy DL model in TRSS to recommend related issues

Continuous improvement with more data and models
Open-source Readiness Success Metrics

- # of contributors to the open-source repos
- # of forks, # of clones
- # of issues reported by external parties
- Speed and ease of adding tests

![Graph showing total git clones and unique git clones over time](image)
Summary - Tips for success

• Actively reduce technical debt or suffer “death by a thousand cuts”

• Make sense of the noise

• Recognize the egos in the room, call out hubris

• Ignore naysayers

• Measure & adjust

• Remember your mission
Contact Information & References

- Adoptium and AQAvit projects:
  - https://projects.eclipse.org/projects/adoptium
  - https://projects.eclipse.org/projects/adoptium.aqavit

- Social media / Slack:
  - Twitter: @ShelleyMLambert
  - https://adoptium.net/slack.html

- Books:
  - “How Google Tests Software” - James A. Whittaker, Jason A. Joseph, and Jeff Carollo
  - “Beautiful Testing: Leading Professionals Reveal How They Improve Software” - Tim Riley