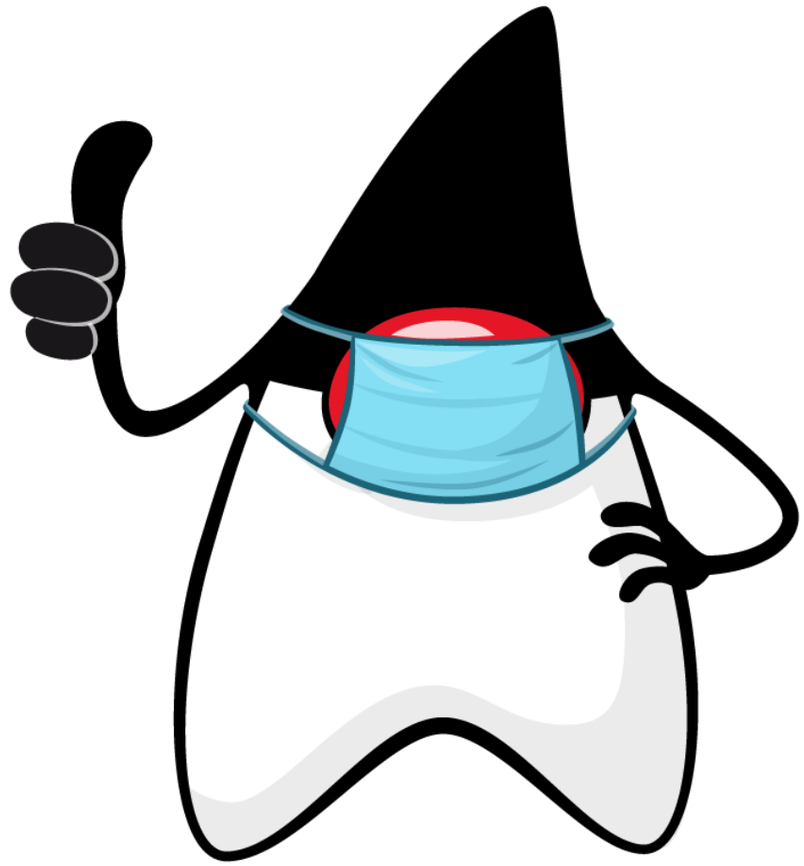
Dismantling Technical Debt and Hubris

@ShelleyMLambert

JPoint 2021

The Method behind the Madness

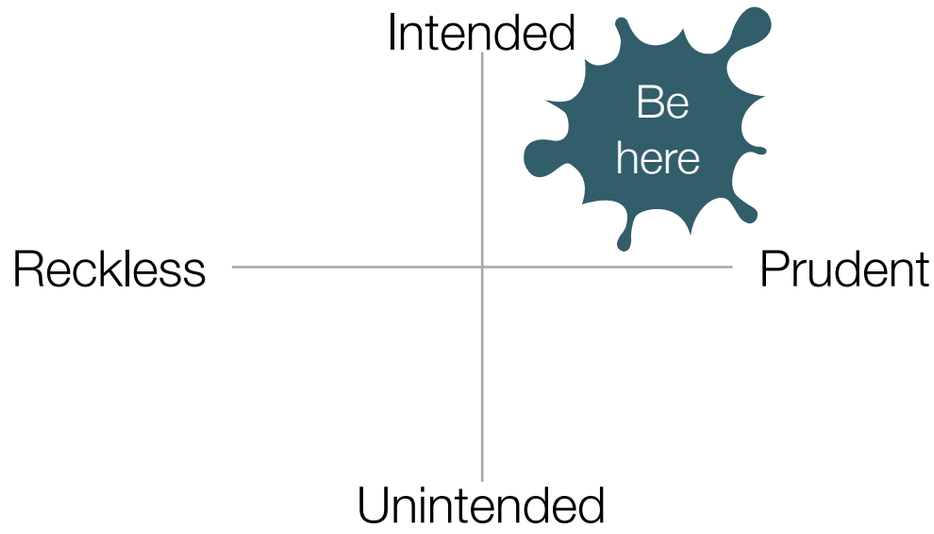
- Definitions
- Various (often conflicting) requests
- 4 modes of activities
    - Maintenance
    - Development Transformation
    - Innovation
    - Open-source Readiness
- Measuring success



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

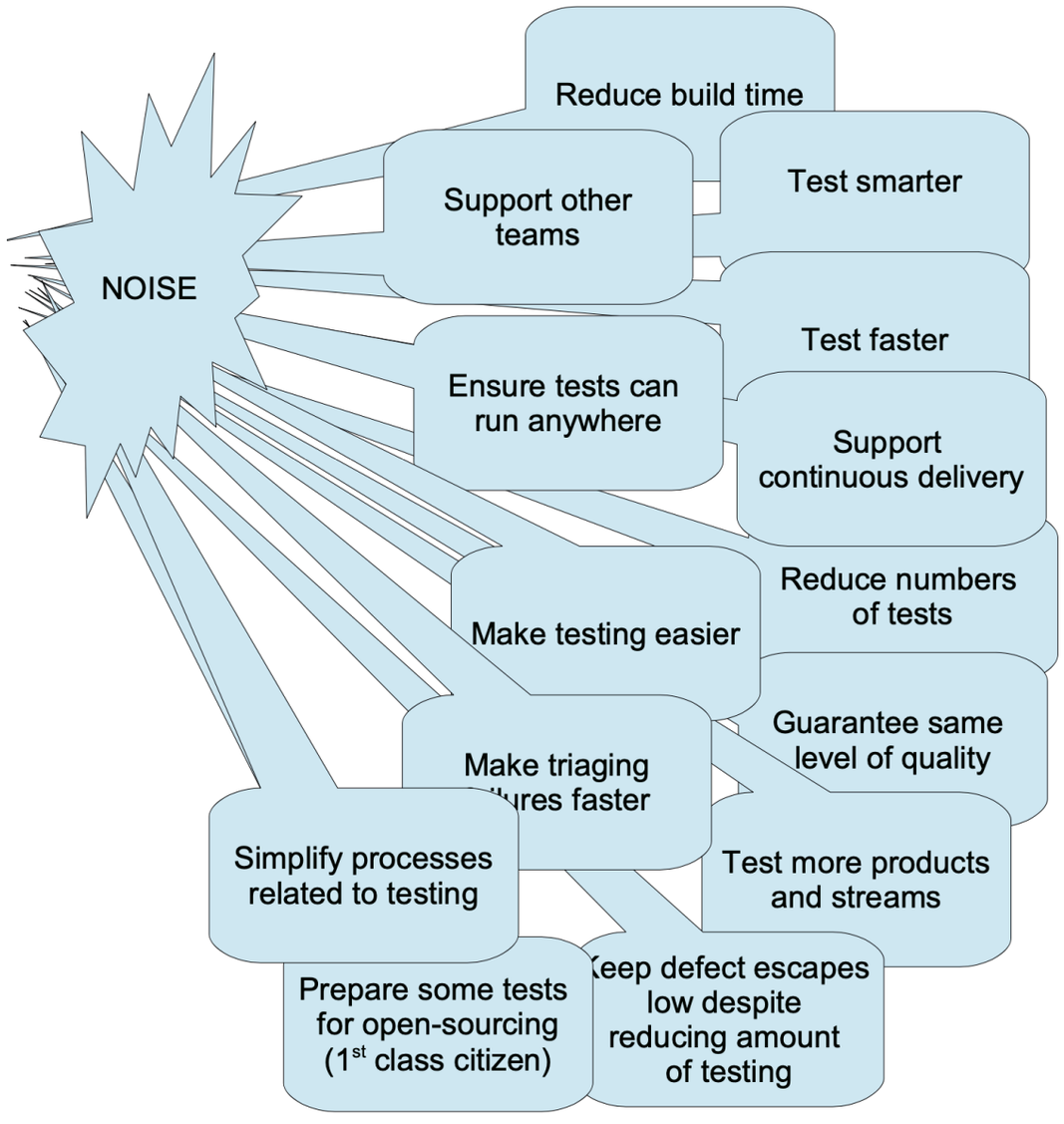


* Technical Debt Quadrant - Martin Fowler

Hubris

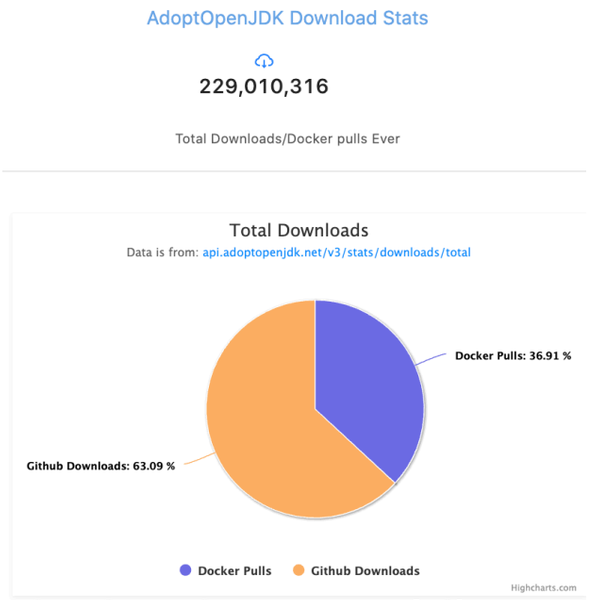
- excessive confidence or arrogance
- leads to lack of questioning and planning, short-sighted, harmful behaviour

The Noise


NOISE
Reduce build time
Support other teams
Test smarter
Test faster
Ensure tests can run anywhere
Support continuous delivery
Reduce numbers of tests
Make testing easier
Guarantee same level of quality
Make triaging
ures faster
Simplify processes related to testing
Test more products and streams
Prepare some tests for open-sourcing (1st class citizen)
eep defect escapes low despite reducing amount of testing


We are not doctors, but we play them
~~on TV~~
at work

Context and Stats



dash.adoptopenjdk.net

Vendor	% in use
Oracle	74.78
AdoptOpenJDK	7.06
IcedTea	5.30
Azul	2.96
IBM	2.37
Amazon	2.18
Unknown	1.96
Pivotal	1.40
SAP	0.74
Sun	0.58
Debian	0.54
Other	0.10

Market share (JDK8, March 2020)

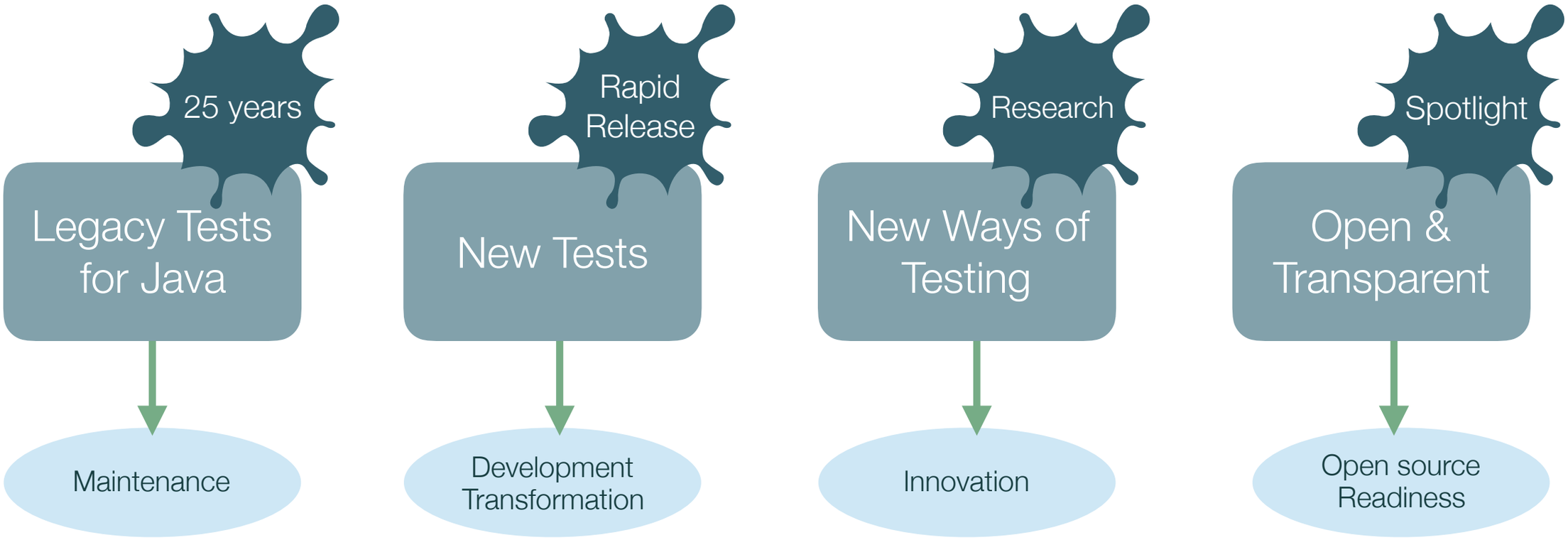
blog.newrelic.com/technology/state-of-java

x 2 JDK8

x 8 JDK11

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

Making Sense of the Noise



- 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	openjdk	system	external	perf	jck
testNG, cmdlinetester	Jtreg, testNG	STF	junit, sbt & others	Assorted benchmarks	javatest

```
$(JAVA_BIN)/java -cp GeneralTest.jar org.testng.TestNG -d $(REPORTDIR) testng.xml -testnames floatSanityTests -groups sanity
```

```
$(JAVA_BIN)/java -Xmx512m -jar jtreg.jar -vmoptions:"-Xmx512m" -w work -r report -jdk:$(JDK_HOME) -exclude:ProblemList.txt jdk_math
```

```
perl stf.pl -test-root=openjdk-systemtest_root -systemtest-prereqs=systemtest_prereqs -java-args=-Xjit -results-root=reportdir -test=ConcurrentLoadTest
```

```
sbt "partest $(TEST_SUITE)" ${ANT_HOME}/bin/ant -Divy_install_path=${ANT_HOME}/lib -lib ${ANT_HOME}/lib -f ${LUCENE_SOLR_HOME}/lucene-solr/build.xml -Duser.home=${LUCENE_SOLR_HOME} -Dcommon.dir=${LUCENE_SOLR_HOME}/lucene-solr/lucene test
```

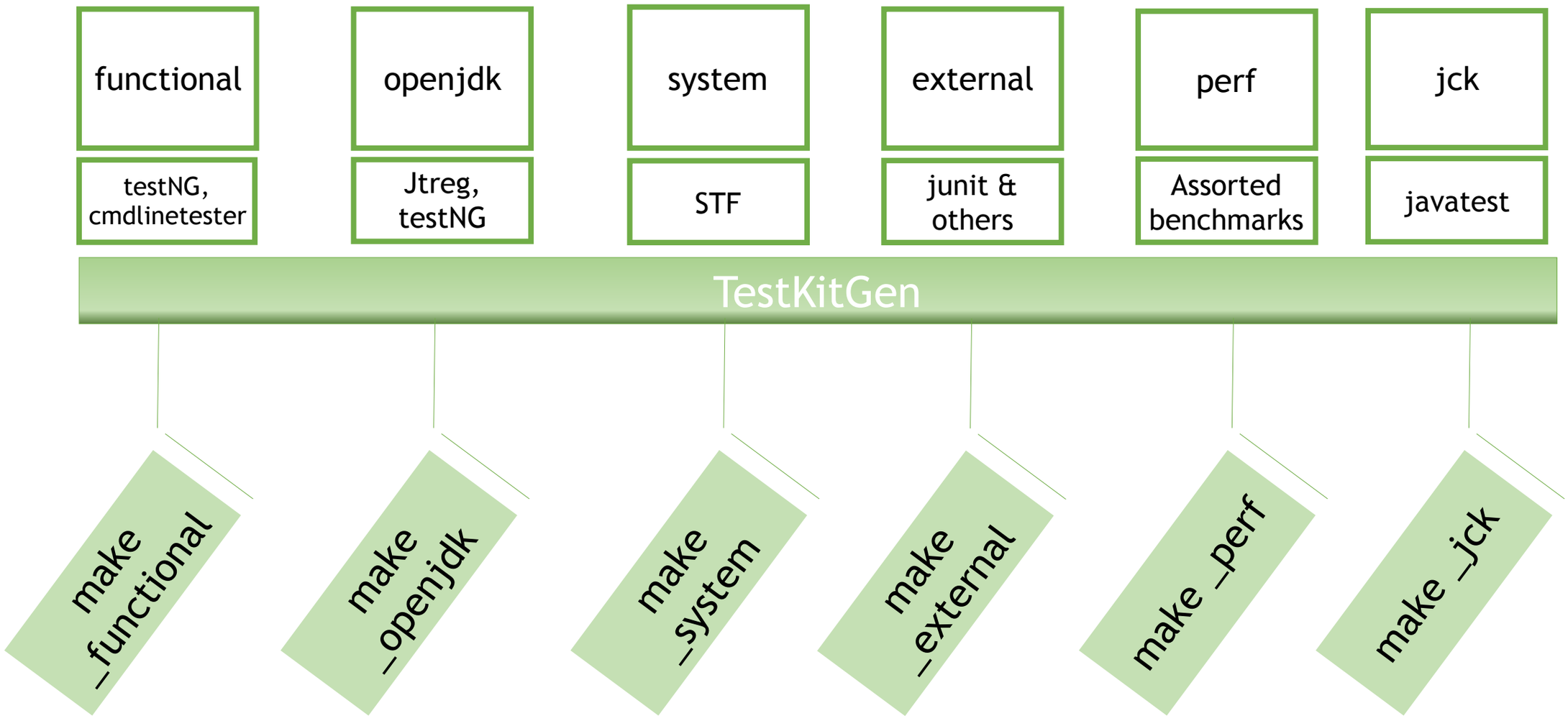
```
$(JAVA_BIN)/java -DBumbleBench.listOptions -jar BumbleBench.jar [Benchmark name]
```

```
some javatest command not to be disclosed...
```

The Problem with too many frameworks

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

Consolidate



3-layer Cake of Automation

Test Results Summary Service (TRSS)

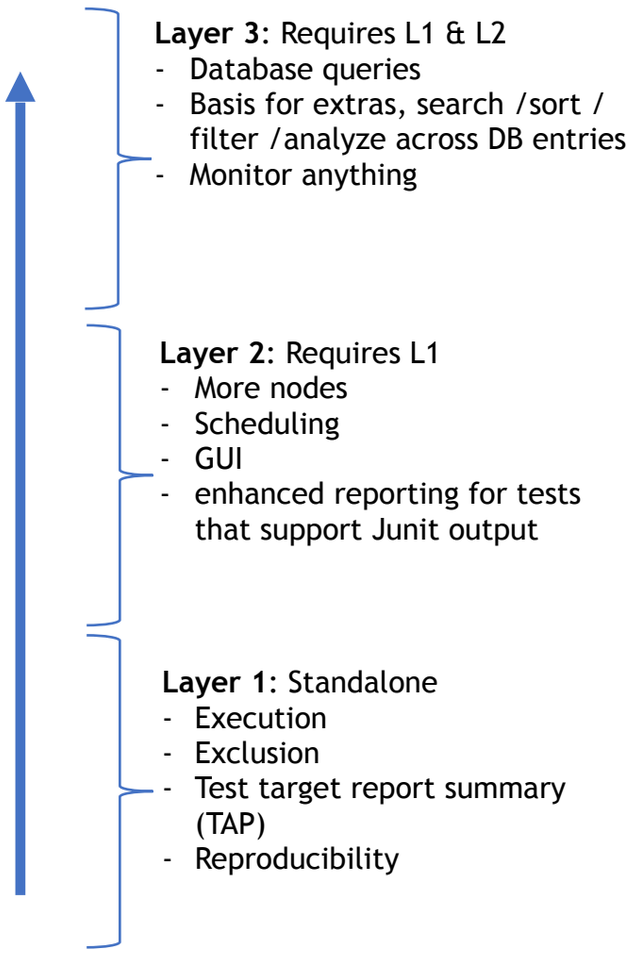
Responsibilities:
- Monitor multiple CI servers
- Graphical, Aggregate summary, Deep history
- Search/sort/filter
- Pluggable parsers
- Basis for deeper analytics and deep learning services

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:
- 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)



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." *

* From "How Google Tests Software"

- **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

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

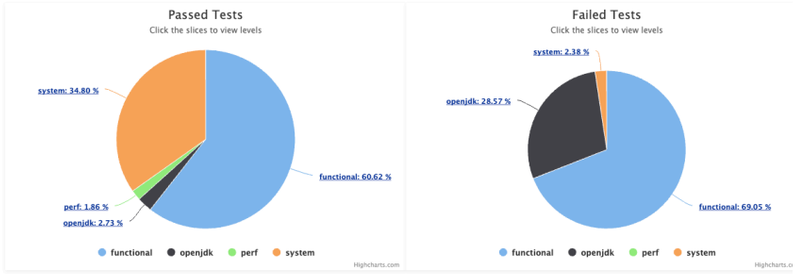
Improve work flow:

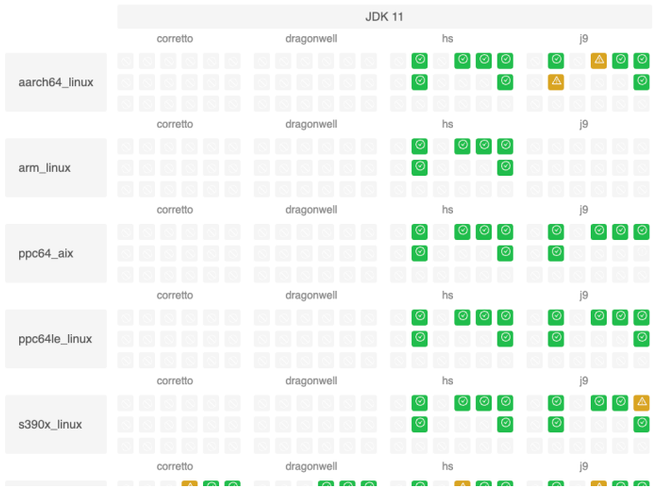
- Remove 'friction', less time per change, less error prone
- Address long-standing legacy problems - tools that created more work (problem tracking)

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?

Mode	Success Metrics
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

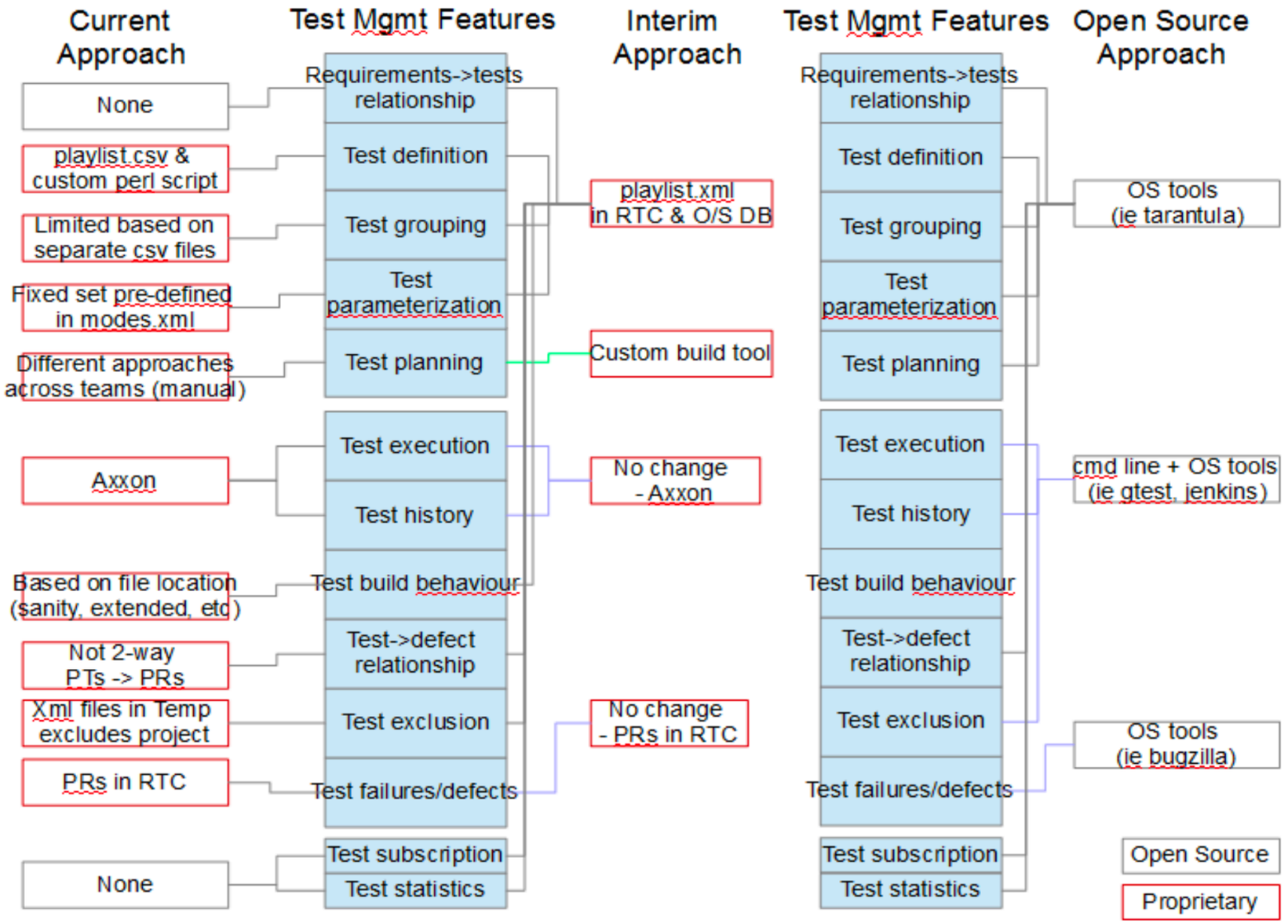
For each suggested metric, what direction is better?

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



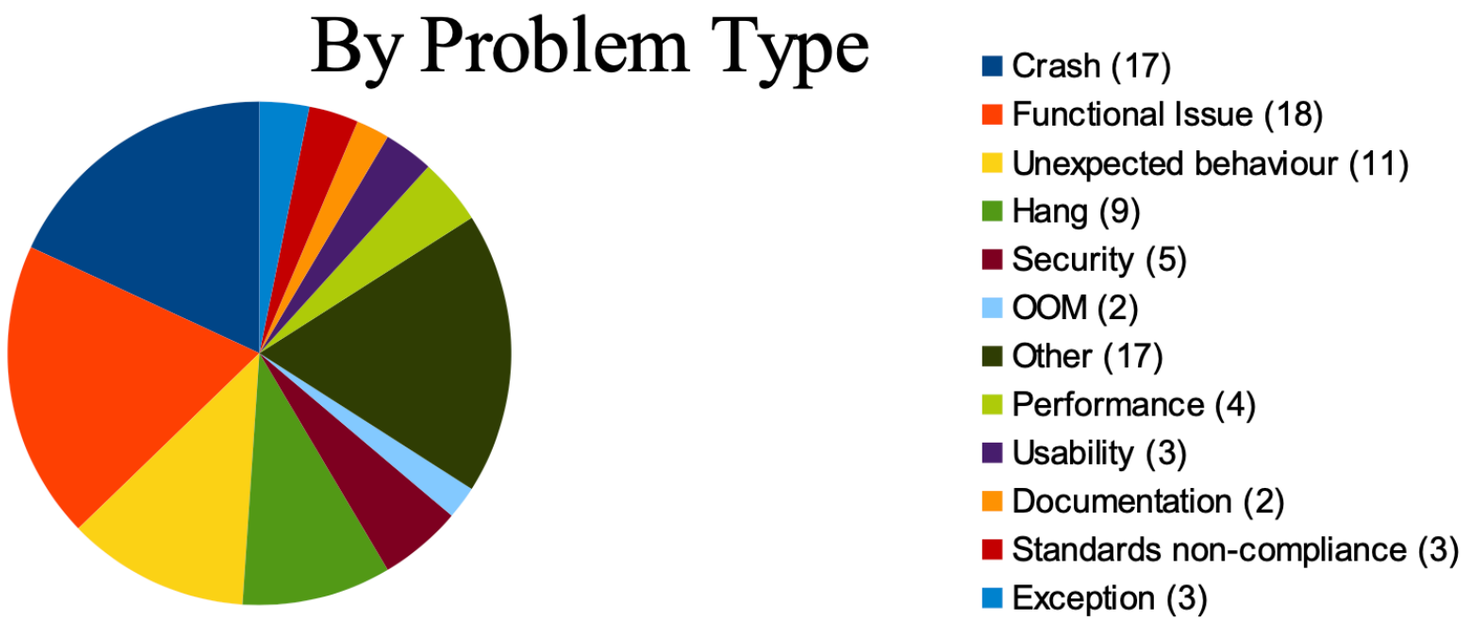
No time for war stories

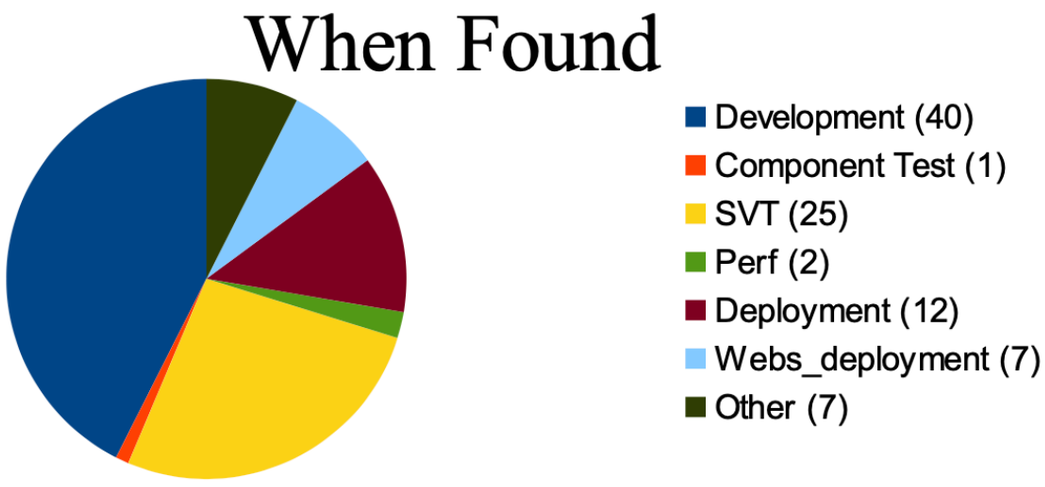
Ask me later!

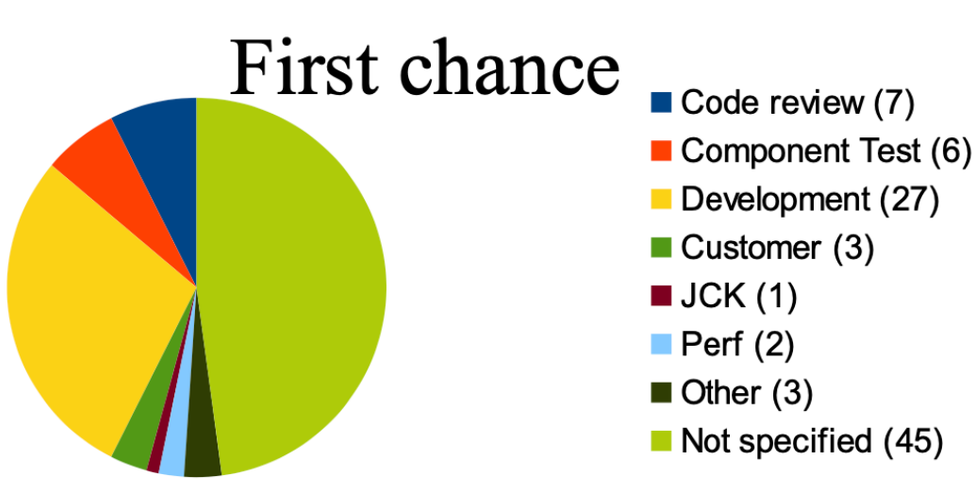
Development Transformation Success Metrics

- 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







TKG example - faster, better, smarter

- Flexible play and granularity
  slice'n'dice via playlists tags

- Standardized output
  enforced TAP output

Additional requirements

On-going refinements

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

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

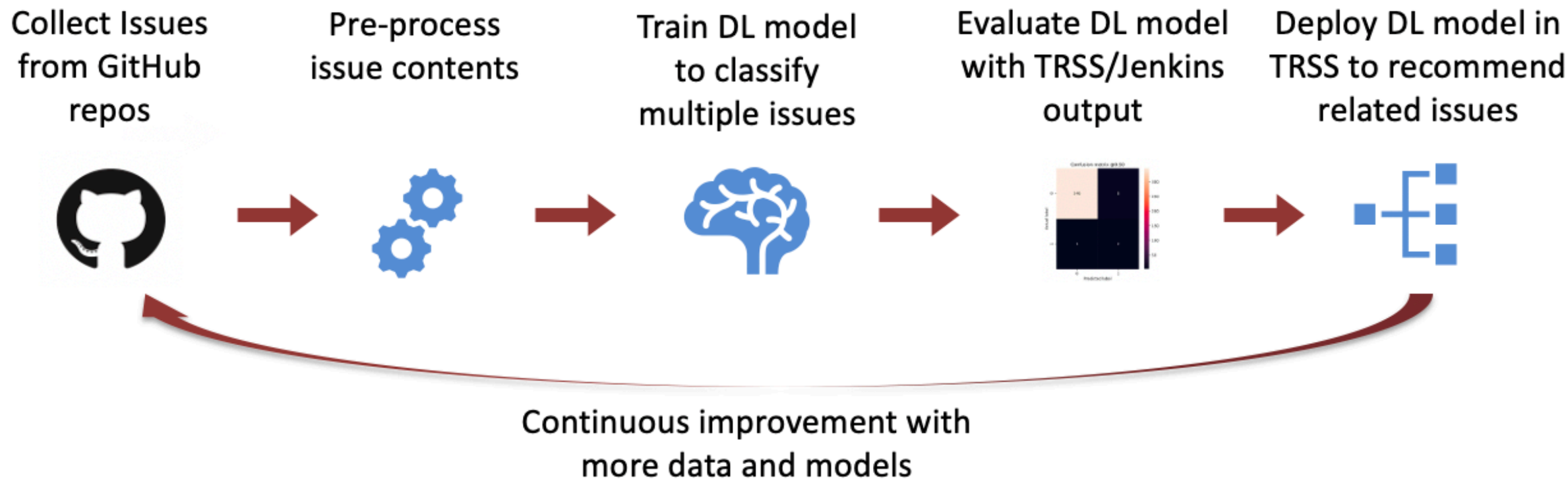
Innovation Success Metrics

- 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

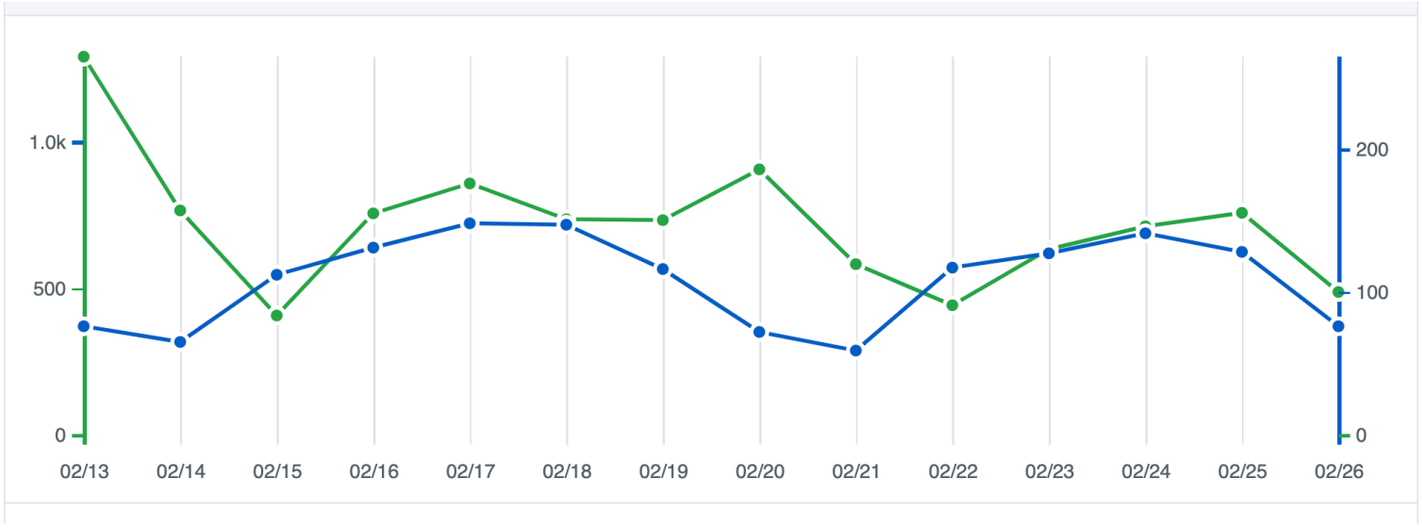
- Utilize deep learning model to recommend possible issues related to test failures

- Github, JBS, StackOverFlow, Support issues



Open-source Readiness Success Metrics

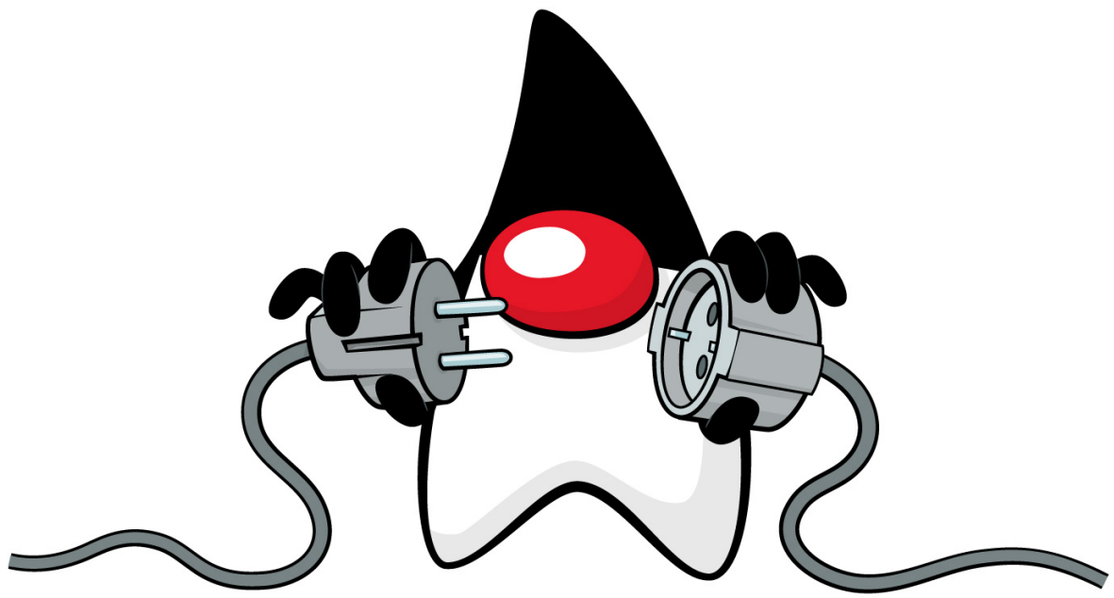
- # of contributors to the open-source repos
- # of forks, # of clones
- # of issues reported by external parties
- Speed and ease of adding tests



Total git clones & Unique git clones

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