Testing methods and tools for large scale distributed systems

Ramon Medrano Llamas
CERN
Crafting (good) software

Life of a distributed system
We were so happy in the 80s…

- One platform.
- One codebase.
- One viewport.
- Release cycles of year(s).

Cray X-MP 48 (ca. 1988)
And then, some day...

- 2.4 billion users
- 566% growth YoY
  - 2000-2012
- Zillions of devices
  - 10 bn by 2016
  - |Mobile| > |Desktop|
  - 260 EiB/year
    - 66 bn DVDs

Source: CERN, W3C, Cisco

"Information Management: A Proposal", Tim Berners-Lee, March 1989
The rise of distributed

- Computer networks vs. distributed systems.
- Latency vs. throughput.
- Batch vs. real-time.
- Highly vs. loosely coupled.
- Committee vs. de facto standard.
- Over architected vs. simple.
Sadly, we...

- **As programmers:**
  - We are not smart enough…
  - We are lazy…
  - We tend to not think globally…
  - We don’t speak with others…
  - We don’t like to learn new stuff…
  - We love politics so damn much…

- Thus, we cannot exploit all the computing resources we have.
- Neither we can make software run reliably
Wait a minute...

- **If we are too bad:**
  - How the LHC can generate aprox. 15 PiB/year?
  - How Gmail can have 425 million users?
  - How Facebook can have 1.01 billion users?
  - How BOINC could averaged 7.28 PFLOPS in 2012?

- Obviously, we can do nifty stuff.
- But we are far, by a long shot, from being efficient.
- Also, quality is a huge concern.

Source: CERN, Google, Facebook
Introduction to software testing

- A bug can be proof, but not the absence of it.
- ...
- Wait…
- I believe that I studied this at the university…
- So, why you don’t test your stuff?
Quality

- **What the f**k is quality?**
  - “Quality can have two meanings:
    a) The characteristics of a product or service that bear on its ability to satisfy stated or implied needs
    b) A product or service free of deficiencies.”

  *American Society for Quality*

- **Quality ≠ Test**
  - “Get it right from the beginning or you’ve created a permanent mess.”

  *James Whittaker*

- **Measurement**
  - “A science is as mature as its measurement tools.”

  *Louis Pasteur*
Quality management

- Quality management must target:
  - Reliability.
  - Efficiency.
  - Security.
  - Maintainability.

- Quality must be measured:
  - Make something subjective objective.
  - If you cannot measure, you cannot optimize.

- Some standard methods:
  - ITIL
  - QFD, 6σ, ERD…
ISO25000: Software QUALity Requirements and Evaluation

2501n: Quality model:
- What is quality?
- External: client’s perception of the product.
- Internal: business process improvements.

2502n: Quality measurement:
- What to measure?
- Mathematical definition of quality.

2504n: Quality evaluation:
- How to measure a software product?
QFD / 6σ / ERD

- **Iterative process improvement:**
  - Statistical, predictive models → minimize variability
  - Introduce quality from the beginning.
  - Release early, release often™
Running a service is complex:
  - How to coordinate suppliers and clients?
  - How to define expected service levels?
  - How to react to change?

ITIL is yet another standard for this:
  - Service Operation.
  - Service Levels.
  - Change management.
Methodologies. Wat?

Life of a bug
A good old friend

A new method

Let engineers be engineers

- No bureaucracy.
- Focus on the user
  - And the rest will follow.
- Good is never enough.

Get Shit Done™

- We love to build stuff.
- Do one thing
  - And do it well.
- And to ship it.

1. Build a perfect team.
2. Build stuff
   1. Plan.
   2. Build.
   3. Go to 1.1.
Test Driven Development

- Sounds harder than it is:
  - Just make tests first…
  - And you get the best contracts for the software.
  - Less coupling and nicer designs.
  - Requires testing automation.

1. Choose a component.
2. Write a test.
3. Make sure that it fails.
4. Fix the stuff.
5. Eliminate redundancies.
The perfect team

- **Feature development:**
  - Owned by software engineers.

- **Unit testing:**
  - Owned by software engineers. Yes.

- **Testing infrastructure and testability:**
  - Software engineers dedicated to this task.
  - Roughly 1 or 2 per team.

- **Test management, analysis and planning:**
  - Test manager (this guy is a software engineer as well).
  - Coordinates all the larger scale testing.
  - Might be the previous role.
The perfect team

- Quality comes from solid engineering
  - Stop talking and go build things.
  - No meetings.

- Don’t hire too many testers
  - Testing is owned by the entire team.
  - Is a culture, not a process.
  - Testers are software engineers. Yes.

- There is not place to mediocrity
  - Hiring good people is the base.
  - And keeping them challenged.
The Agile wave

- It is just a formalization of some old principles:
  - We are humans.
  - Software that works.
  - Client matters.
  - Change happens.

- Look at:
  - SCRUM,
  - Kanban,
  - Lean manufacturing.

- Also
  - Code and fix.
SCRUM

- **Incremental development:**
  - Backlogs.
  - Scrum Master vs. Product Owner.
  - Sprints:
    - Produce something **usable**.

- **Testing in this scenario:**
  - Can be interleaved in each sprint.
  - Integration tests should be built incrementally.
  - Regression tests are fundamental.
Tools for sprinting

- **Post-it + whiteboard:**
  - Very simple.
  - More powerful than one might expect.

- **JIRA + Greenhopper:**
  - CERN’s choice.
  - Integrated with issue collection and SCM.
  - Custom workflows allow the integration of test and review.
  - Integration with Bamboo → Automatic deployments.

- **Microsoft Project:**
  - There is a plugin for SCRUM.
How to nail testing?

- **Attribute, Component, Capability.**

- **This is a method to create:**
  - The design document.
  - Test plan.
  - Risk analysis.

- **In one process of 30 minutes:**
  - Should be done in a quick meeting.
  - Ideally, integrated in the backlog of the product.
  - You just need a spreadsheet for this.
About the Design Document

- It is the encyclopedia of your software:
  - Involve testing and testers from the beginning.
  - Just do this document.
  - Avoid prose, this is not a novel:
    - Just bulleted lists and tables.
  - Make it worth to keep it updated.
Attributes, Components, Capabilities

- An attribute:
  - Sales people and managers have them.
  - They are… attributes.

- e.g. A batch system:
  - Reliable.
  - Easy.
  - Generic.
  - Scalable.
Attributes, Components, Capabilities

- A component:
  - Engineers have them.
  - They are nouns.

- e.g. A batch system:
  - Scheduler.
  - Worker node.
  - Job.
  - Queue.
Attributes, Components, Capabilities

- **Capabilities:**
  - A subset of the A×C product.
  - They are verbs.
  - Looks like a nice description of requirements.

- e.g. A batch system:
  - {Generic, Worker node}: Enables different types of jobs in each WN.
  - {Easy, Job}: Users are abstracted of the infrastructure.
  - {Reliable, Scheduler}: Supports n failures on the infrastructure.
  - {Scalable, Queue}: Allows O(x) users in parallel.
  - ...

---

Testing on large scale distributed systems

---

iCSC2013, Ramon Medrano Llamas, CERN
Attributes, Components, Capabilities

- **Risk analysis:**
  - You will test capabilities.
  - But not everything at once.
  - Add:
    - Impact ∈ {Minimal, Some, Considerable, Fatal}
    - Frequency ∈ {Rarely, Seldom, Occasionally, Often}
    - And you have a risk assessment.

\[ Risk(a, c) = \sum_{C} Impact(a, c, C) \cdot Frequency(a, c, C) \]
### Attributes, Components, Capabilities

<table>
<thead>
<tr>
<th></th>
<th>a × c = C</th>
<th>Scheduler</th>
<th>Worker node</th>
<th>Job</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable</td>
<td>19</td>
<td>6</td>
<td>9</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>2</td>
<td>3</td>
<td>15</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Generic</td>
<td>9</td>
<td>15</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Scalable</td>
<td>18</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
**Case study: AliEn mocking**

- **Implements the full submit queue:**
  - Every changeset triggers a build
  - And a test pass.

- **The interesting thing at this point is in the mocking:**
  - The systems sets up a entire grid in a node for testing:
    - A Storage Element
    - A Computing Element
    - Even central services.
Legacy management

- **Legacy:**
  - Has its value.
  - Cost is rising quickly.
  - People don’t know how it works.

- **It is not always easy:**
  - Try to move things up.
  - Virtualization can help…
    - Don’t stack shit if possible.
  - Pay the technical debt:
    - Integrate on test strategies.
    - Integrate on deployment systems.
Test certification

- How do you progress on testing:
  - Set some milestones in the form of levels.

- Level 1:
  - Continuous integration in place.

- Level 2:
  - Pre-commit smoke testing.

- Level 3:
  - All features covered by integration tests.

- Level 4:
  - No-nondeterministic testing.
Test automation

Life of a changeset
Test automation

A testing infrastructure must:

- Allow developers to get a unit test results immediately.
- Allow developer to run all unit tests in one go.
- Allow running tests for only the changed components.
- Allow code coverage calculations.
- Allow running unit tests for the changesets submitted.
- Show dashboards of testing evolution.
Test types

- **Unit test:**
  - Executes in less than 100 ms.
  - Test just a module.

- **Integration tests:**
  - Executes in less than 1 minute.
  - Test interaction between few modules.

- **System tests:**
  - Execute as quickly as possible.
  - You can test a functionality with mock data.
Developing code

1. Build the feature in a set of files that compile.
2. Build it as library target.
3. Write unit tests for that.
4. Automate the unit testing
   1. A make rule…
   2. A CI profile…
5. Run static analysis tools.
6. Now you can define a binary target and run the thing.
7. Send the changeset for review.
Pre-commit

- **SCM is not optional:**
  - It is the base of any test automation.
  - Subversion vs. Git.
    - They are both good.
    - No flames, please.

- **The pre-commit checks:**
  - A subset of the unit tests (“smoke tests”).
  - Never allow a commit without smoke compliance.
The Code Review

- The code review is well established in large companies:
  - Other’s view on your work.
  - Maximize readability.
  - Keeps code reusable and self-contained.
Code review tools

- **Rietveld:**
  - A version of Mondrian.
  - Couples with Subversion.

- **Gerrit:**
  - Written in Java.
  - Couples with Git.

- **GitHub’s pull requests:**
  - A distributed model based on Git.

- **More:**
  - Review Board, Mavelich
Submit queue

- The submit queue runs all the tests in each commit:
  - Starts on a clean working copy.
  - Applies the changeset.
  - Runs the tests.
  - All of this in parallel with other change submissions.

- If everything green:
  - Deploy.
Release

- Use different release channels:
  - Canary
    - Nightly builds, for engineers and testers.
  - Alpha and Beta
    - For engaged users.
    - Make feedback submission easy for them!
  - Stable
    - Release for production.

- This pipeline has to be autonomous.
Continuous Integration

- A CI system helps implementing this:
  - Jenkins
  - Travis
  - Bamboo
    - Integrated with JIRA & Co.
    - Interesting for CERN’s case.

- GitHub’s HUBOT:
  - Control all your procedures on the chat.
  - Set of CoffeScript scripts to interface services.
  - From builds to deployments.
Case study: LHCbDirac

- Using Jenkins for each build
  - Automated build with dependency detection

- Passes tests automatically
  - After build succeeds

- Static analysis through Coverity
  - Automatic test coverage analysis
  - And static analysis checks
Test analysis

- **Who to blame in case of bug:**
  - The entire team has failed.

- **Statistics:**
  - Detect background bugs.
  - Code coverage analysis.

- **Provide dashboards:**
  - Needed to control the automation.
Case study: CORAL/COOL

- **Functional testing:**
  - Automatic unit testing
  - On production platforms
    - GNU/Linux + OS X

- **Non-functional testing:**
  - Performance probes against the Oracle DB
  - Code coverage measured with Coverity
Compatibility labs
UX testing automation

- **Should be automated:**
  - In order to be regressed.

- **Selenium:**
  - JS testing on the browser.

- **WebDriver:**
  - A proposal of a API to automate testing.

- **Bots:**
  - Google’s approach for validation of Chrome.
Crowd sourcing

- **Release a beta version:**
  - People love them.
  - Make easy to send feedback.
  - Instantly, free testing:
    - Real environments,
    - real use cases,
    - more difficult to repeat.

- Use the proper channels.

- **Amazon’s Mechanical Turk**
  - Used by Twitter for search quality.
Autonomic Computing

Life of an issue
Autonomic systems

- If every time things break the same way:
  - Just make the system auto recover.
  - If you run on an IaaS, you have much already done.

- **Autonomic Computing:**
  - A vision from IBM Research.
  - 2003!

- **Systems:**
  - Self-configure.
  - Self-heal.
  - Self-optimize.
  - Self-protect.
Self-configure

- Automatic application of configurations.
- Automatic configuration of software.
- Automatic integration in network environments.

- Just set some policies and the rest follows.
Self-heal

- Automatic recovery of failures.
- Automatic fail-over on upgrades.
- Automatic diagnose.
- Systems react to problems automatically.
Self-optimize

- Automatic performance tuning.
- Automatic clean up.
- Automatic update.

- Learning and monitoring for machines, not humans.
Self-protect

- Automatic protection against large-scale attacks.
- Anticipation of problems.
- Not only security issues, but failures of components…
The orchestrator

Autonomic manager

Policies

Know-how

Sensors

Actuators

Managed element
Case study: The Agile Infrastructure

- Renovation of tools at CERN:
  - Go with the industry…
  - If they manage $O(1,000,000)^*$ machines, why not us?

- A case of study for operations and development.

- A step closer to an autonomic system.

- Automation of operations via:
  - Self-configuration.
  - Self-optimization.

- In theatres this spring.

* The Big O notation is clearly wrongly used here, but you get the idea.
The DevOps way

- **Usually, there was a problem:**
  - Developers and operations were two like water and oil.

- **DevOps shifts responsibilities:**
  - Built quality into product first.
  - The team is responsible of project success, not some areas.
  - Break down barriers between departments.
  - Remember the Test-enabled team?

- **Requirements: CAMS**
  - Culture
  - Automation
  - Measurement
  - Sharing
Implications

- Reduced changes, more often.
- Tighter collaboration between stakeholders.
- Less risk on each deployment.
  - Ideally, there are no deployments.
  - Makes the Autonomic Process closer.
- Developers are in control. And operators.
Testing on large scale distributed systems

Tools for Culture Change
Tools for Self-configuration

- **Puppet:**
  - Very high momentum.

- **Chef:**
  - Very similar to Puppet.

- **The Foreman:**
  - A frontend for Puppet.
  - Can provision bare metal and cloud resources.

- **Marionette Collective:**
  - Orchestrator tool for Puppet.
OpenStack

- Is a cloud engine (IaaS).

- **Started by RackSpace:**
  - Managing 79,000 machines as of 2011.

- **Offers:**
  - Compute: nova.
  - Object storage: swift, cinder.
  - Virtual networking: quantum.
  - Many more modules.

- **CERN’s choice.**
  - And NASA’s, HP’s, Wikimedia’s, Canonical’s, Intel’s…
  - There are other options: CloudStack, OpenNebula…
Tools for measurement

- Puppet reports.
- Splunk, New Relic, Logstash...
- Graphite:
  - Integrate monitoring data from different sources.
- Old friends:
  - Nagios, Ganglia.
- But #monitoringsucks.
  - Clunky interfaces
  - Host centric
  - Tied hands
Case study: perfsonar

- **While accessing data:**
  - Collect metrics of network performance.
  - Deploy point-to-point monitoring.
    - `iperf`
  - Detect slowness
  - Allows the introduction of Software Defined Network
    - Self-optimization!
Live testing

Life of a distributed system (2)
Live testing

- **Testing does not end on release:**
  - These systems need to be monitored,
  - and checked for availability.
  - The only way to do it is with active testing.
  - And it is a requirement of self-healing.
  - And can play a big role in self-optimization.

- **Approximations:**
  - Profiling, probing.
  - Monitoring.
  - Either case: be careful! You are playing in production.
System wide profiling

- **Profile everything on production:**
  - There is an overhead of profiling.
  - But it is worth.

- **Procedure:**
  - Instrument the software to emit performance metrics.
  - Collect them and start gathering statistics.
  - Apply machine learning techniques to predict.
Probing

- **Simulate the end user:**
  - Send real jobs and look
  - This is a kind of system testing,
    - But on real infrastructure
  - Collect data as on SWP.
  - Apply the same techniques for data mining.
#monitoringsucks

- **Monitoring is so old-fashioned:**
  - You have to look at stuff.
  - Let the system work for you.

- **The true power of SWP comes:**
  - When the machines are able to auto tune.
  - When, based on load, the infrastructure changes:
    - To cope with it,
    - To save power,
Case study: DDM autoexclusion

- Excludes storage sites based on:
  - Metrics of free space available: SRM spacecollector.
  - Scheduled downtimes from AGIS.
  - SAM testing (Nagios).

- With this, it produces exclusions automatically:
  - No operators needed.
  - Self-configuration!
Case study: HammerCloud

- It can be submitted to something?
  - Then it can be tested with HammerCloud
  - Mainly batch systems and IaaS.

- Stress testing:
  - Measure the resources
  - Analyse,
  - Tune
  - (Looks like QFD…)

- Functional testing
  - The live testing.
HC’s Functional testing

- **Monitors:**
  - Availability
  - Functional quality
  - User-seen performance.

- **HC is always submitting probes everywhere.**
  - And collects data.
  - 50,000,000 probes/year
  - Offers monitoring
  - And actions.
HC’s Infrastructure probing

- HC can test anything with an API.

- For instance, cloud resources:
  - VM life cycle duty.
  - Site performance:
    - Network,
    - CPU overhead.

- Tested resources on several clouds:
  - T-Systems, CloudSigma, Atos, RackSpace, Google…
  - More than 40,000 CPU hours.
Testing distributed systems

Ramon Medrano Llamas
CERN
References

- The Art of Software Testing
  - Glenn Myers

- Testing Computer Software
  - Cem Kaner

- Test-Driven Development by Example
  - Kent Beck

- How Google Tests Software
  - James Whittaker, Jason Arbon and Jeff Carollo
Acknowledgments

- José García Fanjul,
  - Professor and researcher on testing
  - University of Oviedo

- All the people at IT-ES for the input
- James Whittaker for awesome books
- Teams at Google for releasing awesome tools
- Teams at Heroku and GitHub for Getting Shit Done™
Case study: Android’s submit queue