Continuous Integration, Delivery and Deployment

Eero Laukkanen
T-76.5613 - Software Testing and Quality Assurance P
20.11.2015
System Integration

In engineering, **system integration** is defined as the process of **bringing together** the component subsystems and ensuring that the subsystems **function together** as a system.

Software Integration

- System Integration
- Change Integration: Merging

Possible integration conflict!
Integration Conflicts

• **Merge Conflicts**
  • Detected by version control software

• **Semantic Conflicts**
  • Detected by compiler, tests and code review

http://martinfowler.com/bliki/SemanticConflict.html
Regression Testing

The purpose of regression testing is to ensure that changes have not introduced new faults.

Experience has shown that as software is changed, emergence of new faults and/or re-emergence of old faults is quite common.

https://en.wikipedia.org/wiki/Regression_testing
History of Software Integration and Regression Testing

- Prehistory: Big Bang Integration
- 1996: Daily Build and Smoke Test (Steve McConnell)
- 2000: XP and Continuous Integration (Kent Beck)
- 2006: Continuous Integration (Martin Fowler)
- 2009: Continuous Deployment (Timothy Fitz)
- 2010: Continuous Delivery (Jez Humble & David Farley)
- 2012: Experiment System (Holmström Olsson et al.)
Big Bang Integration

- **System Integration**
  - Develop subsystems independently, integrate when ready
- **Change Integration**
  - Develop changes in branches, merge when ready
Big Bang Integration

“We entered a huge depressing warehouse stacked full with cubes. I was told that this project had been in development for a couple of years and was currently integrating, and had been integrating for several months. My guide told me that nobody really knew how long it would take to finish integrating. From this I learned a common story of software projects: integration is a long and unpredictable process.”

http://www.martinfowler.com/articles/continuousIntegration.html
Big Bang Integration

- **Development** is simple, because *nothing is changing*
- **Integration** is complex and takes *unpredictable time*
Daily Build and Smoke Test

Prospecting for programmer’s gold.

If you want to create a simple computer program consisting of only one file, you merely need to compile and link that one file. On a typical team project involving dozens, hundreds, or even thousands of files, however, the process of creating an executable program becomes more complicated and time consuming. You must “build” the program from its various components.

A common practice at Microsoft and some other shrink-wrap software companies is the “daily build and smoke test” process. Every file is compiled, linked, and combined into an executable program every day, and the program is then put through a “smoke test,” a relatively simple check to see whether the product “smokes” when it runs.
1996: Daily Build and Smoke Test

- Build and smoke test the whole software system daily
- Fix broken builds
- “Integrate… usually once every few days”

Daily Build and Smoke Test

- Minimizes integration and low quality risk
- Easier defect diagnosis
- Improved morale

![Graph comparing Big Bang and Daily build]

- Big Bang:
  - Vertical axis: code
  - Horizontal axis: time
  - Risk is shown as a vertical line.

- Daily:
  - Continuous integration
  - More developed code and verified code over time
Daily Build and Smoke Test

- Context of large systems
- Tests allowed to take multiple hours
- Timed builds
- Smoke tests evolve when the system evolves
- Dedicated roles take care of the build
Daily Build and Smoke Test

“The smoke test should be **thorough enough** that if the build passes, you can assume that it is stable enough **to be tested more thoroughly**.”

“The standard needs to set a quality level that’s strict enough to **keep showstopper defects out** but lenient enough to **disregard trivial defects**, an undue attention to which could **paralyze progress**.”
2000: Extreme Programming

- Holistic software development methodology
- Continuous Integration, one of the many practices

Extreme Programming and Continuous Integration

“No code sits unintegrated for more than a couple of hours.”

“At the end of every development episode, the code is integrated and all the tests must run at 100%.”

“You need a reasonably complete test suite that runs in a few minutes.”
Continuous Integration

Develop (a couple of hours) \rightarrow Integration & Testing (a few minutes)

\text{pass} \quad \text{fail}

Fix broken build
2006: Continuous Integration Extended

- Single source repository
- Integration machine
- Fix broken builds immediately
- Keep the build fast (10 minutes)
- Test in a clone of the production environment
- Automate deployment

http://www.martinfowler.com/articles/continuousIntegration.html
2009: Continuous Deployment

“The high level of our process is dead simple: **Continuously integrate** (commit early and often). On commit automatically **run all tests**. If the tests pass **deploy to the cluster**. If the deploy succeeds, repeat.”

http://timothyfitz.com/2009/02/08/continuous-deployment/
http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/
Continuous Deployment

“So what magic happens in our test suite that allows us to skip having a manual Quality Assurance step in our deploy process? The magic is in the scope, scale and thoroughness. It’s a thousand test files and counting. 4.4 machine hours of automated tests to be exact.”

“Great test coverage is not enough. Continuous Deployment requires much more than that. Continuous Deployment means running all your tests, all the time. That means tests must be reliable.”
Continuous Deployment

• Context of cloud systems
• Problems in production will always happen
• How to mitigate the problems?
  • Smaller releases have smaller scope and are easier to debug
  • Automated deployments allow fast fixing
  • Deploy to a subset of users first to mitigate problem scope
• Production can be monitored and reverted automatically
Continuous Deployment

- Minimizes release risk
- Easier defect diagnosis
- Improved morale

Frequent verification

Frequent release

Risk

developed
verified
released
2010: Continuous Delivery

• Every change should be releasable…
• ...but not necessarily released automatically
• Allows human verification

Continuous Delivery

Deployment Pipeline

- Increasing confidence in build's production readiness
- Environments become more production-like
- Faster feedback

- Commit stage: Compile, Unit test, Analysis, Build installers
- Acceptance test stage
- User acceptance testing
- Capacity testing
- Production
Continuous Delivery

Commit stage
- Compile
- Commit tests
- Assemble
- Code analysis

Acceptance stage
- Configure environment
- Deploy binaries
- Smoke test
- Acceptance tests

Operations
- Perform push-button releases

UAT
- Configure environment
- Deploy binaries
- Smoke test

Capacity stage
- Configure environment
- Deploy binaries
- Smoke test
- Run capacity tests

Production
- Configure environment
- Deploy binaries
- Smoke test

Artifact repository

Developers
See code metrics and test failures

Testers
Self-service deployments
Continuous Delivery

You’re doing continuous delivery when:

• Your software is **deployable** throughout its lifecycle
• Your team **prioritizes** keeping the software deployable over working on new features
• Anybody can get fast, **automated feedback** on the production readiness of their systems any time somebody makes a change to them
• You can perform **push-button deployments** of any version of the software to any environment on demand

[http://martinfowler.com/bliki/ContinuousDelivery.html](http://martinfowler.com/bliki/ContinuousDelivery.html)
Experiment System

“actual deployment of software functionality is seen as a way of experimenting and testing what the customer needs”

• Beyond regression testing and integration
• Avoid software bloat
Delivery Activities

Develop
Integrate
QA
Release
## Waterfall Process

<table>
<thead>
<tr>
<th>Develop</th>
<th>Integrate</th>
<th>QA</th>
<th>Release</th>
</tr>
</thead>
</table>

**Example Diagram:**

- **Develop**
- **Integrate**
- **QA**
- **Release**
Continuous Integration

| Develop + Integrate | QA | Release |
Continuous Delivery

Develop + Integrate + QA

Release
Continuous Deployment

Develop + Integrate + QA + Release
Practitioner surveys

• 2015 State of DevOps Report
  • ~5000 respondents
• 2013 Continuous Delivery: A Maturity Assessment Model
  • ~300 respondents

http://info.thoughtworks.com/Continuous-Delivery-Maturity-Model.html
Survey: deployment frequency

Distribution of deployment frequency by performance cluster

- Super High
- High
- Med
- Low

- <15 min
- 1h-1d
- 1w-1m
- 1d-1w
- 6m+
Survey: deployment lead time

Distribution of deployment lead time by performance cluster
Survey: practices

“Please rate your development organization’s frequency of use of the following software development practices on a scale of 1 to 5.”

- The development team manages source code using a source code management (SCM) or version control tool
  - Daily/on demand: 40%
  - Weekly: 21%
  - Monthly: 18%
  - Less than monthly: 12%
  - Never: 7%

- The development team uses continuous integration processes to make sure source code is frequently integrated into a common baseline
  - Daily/on demand: 29%
  - Weekly: 28%
  - Monthly: 21%
  - Less than monthly: 12%
  - Never: 7%

- The development team or QA organization uses automated testing to verify quality
  - Daily/on demand: 30%
  - Weekly: 23%
  - Monthly: 24%
  - Less than monthly: 16%
  - Never: 7%

http://info.thoughtworks.com/Continuous-Delivery-Maturity-Model.html
Survey: barriers

“When it comes to releasing your applications/systems/products on a more frequent basis, how much of a barrier are the following items (on a scale of 1 to 5)?”

<table>
<thead>
<tr>
<th>Barrier</th>
<th>1 (Not a barrier)</th>
<th>2</th>
<th>3 (Partial barrier)</th>
<th>4</th>
<th>5 (Critical barrier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A lack of slack time for continuous improvement</td>
<td>6%</td>
<td>12%</td>
<td>32%</td>
<td>30%</td>
<td>18%</td>
</tr>
<tr>
<td>Restricted windows for releases due to ongoing business operations</td>
<td>7%</td>
<td>16%</td>
<td>30%</td>
<td>29%</td>
<td>15%</td>
</tr>
<tr>
<td>Releases cause too many unforeseen issues that need to be managed or require rollbacks</td>
<td>8%</td>
<td>26%</td>
<td>28%</td>
<td>27%</td>
<td>10%</td>
</tr>
<tr>
<td>Lack of appropriate people with necessary skills to speed up releases</td>
<td>11%</td>
<td>16%</td>
<td>38%</td>
<td>21%</td>
<td>13%</td>
</tr>
<tr>
<td>Annual budgeting or accounting processes prevent us from releasing more frequently or changing scope</td>
<td>15%</td>
<td>13%</td>
<td>37%</td>
<td>18%</td>
<td>14%</td>
</tr>
<tr>
<td>Unavailability of testing environments prevent us from quickly assessing releases</td>
<td>19%</td>
<td>15%</td>
<td>35%</td>
<td>20%</td>
<td>10%</td>
</tr>
<tr>
<td>Concerns around auditing and compliance</td>
<td>15%</td>
<td>22%</td>
<td>35%</td>
<td>17%</td>
<td>9%</td>
</tr>
</tbody>
</table>

http://info.thoughtworks.com/Continuous-Delivery-Maturity-Model.html
Problems when adopting continuous delivery

- Build design
  - Complex build, inflexible build
- Product design
  - Modularization, internal dependencies, database schema changes, unsuitable architecture
- Integration
  - Large commits, broken build, merge conflicts, work blockage, long-running branches...
- Testing
  - Ambiguous test result, flaky tests, untestable code, time-consuming testing, UI testing...
- Release
  - Customer data preservation, documentation, feature discovery, marketing, more deployed bugs
- Human & Organization
  - Lack of discipline, more pressure, lack of motivation, lack of experience, team coordination...
- Resources
  - Effort, insufficient hardware resources, network latencies

Problems are connected

What to consider

- Value of fast release cycle
  - Competitive market
  - Product ambiguity
- Cost of fast release cycle
  - Competences, legacy code, infrastructure, maintenance
- How often can you release
  - Cost of release
- How long can quality assurance take
  - Cost of production bugs
Thanks!
eero.laukkanen@aalto.fi