Continuous Delivery Workflow for Accelerator Control Software

Feedback, Lessons Learned and Next Steps

Jean-Christophe Garnier, CERN
On behalf of the Machine Protection Software Team
With special thanks to Kamil Król
Introduction

● Software is everywhere
● Continuously changing, evolving
  ○ Security updates
  ○ Adapt to external dependency changes
  ○ Bug fixes
  ○ New features
● Personal perspective on operational software upgrades?
  ○ 5 hours of beam lost due to bug introduced with a single software upgrade early 2017

● Are transparent software upgrades possible?
● Can operational software be deployed in a way that minimizes downtime?
Outline

1. Software environment
2. Continuous Delivery
Software Project - the example of AccTesting

- Commissioning of electrical circuits and machine protection systems
  - Up to 23000 tests after the first long shutdown, on ~2000 circuits
- Full automation of test scheduling and execution
  - Following defined constraints to avoid conflicts
- >50% of test result analysis is now automated
  - Consistent and fast analysis of circuit performance
  - Increasing this share every year
- Next steps: integrate more systems e.g. BLMs, interlocks, etc.
10 products, all related
>1 Million lines of code
Software Team

● Unstable team
  ○ From 5 to 12 members
  ○ New ideas, technologies and methodologies
  ○ Domain expertise loss

● Challenges
  ○ Growing complexity of projects
  ○ Prevent quality degradation
  ○ How to grow and maintain Institutional Memory?

● Agile Scrum by the book since July 2012
Software Challenges

- Maintain legacy code
  - Address bug fixes
  - Adapt to external dependencies

- Adding value
  - Address requirement changes
  - Develop new software tools
    - New devices
    - Ad hoc online monitoring framework
    - Public data service

Brian Foote
Joseph Yoder
“Big Ball of Mud”

ARW 2017, Jean-Christophe Garnier, CERN
Software Challenges - Defects

Study from the Systems Sciences Institute at IBM

➢ Value of iterative processes

<table>
<thead>
<tr>
<th>Software Development Phases</th>
<th>Percent of Defects Introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>20 %</td>
</tr>
<tr>
<td>Design</td>
<td>25 %</td>
</tr>
<tr>
<td>Coding</td>
<td>35 %</td>
</tr>
<tr>
<td>User Manuals</td>
<td>12 %</td>
</tr>
<tr>
<td>Bad Fixes</td>
<td>8 %</td>
</tr>
</tbody>
</table>

Source: Computer Finance Magazine

ARW 2017, Jean-Christophe Garnier, CERN
Continuous Delivery
Deployment the good old way

- Tedious
- Error prone
- Lengthy

Simplified dependency tree of AccTesting Server

ARW 2017, Jean-Christophe Garnier, CERN
Wishlist for Continuous Delivery

- Fast delivery of features, requirements changes and bug fixes
- Focus on valuable work
- Confidence in the software we deploy
  - From ourselves, developers
  - From our users
- 1 click deployment and 1 click rollback
Continuous Delivery Definition

“Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.”

Martin Fowler (2013)

“We follow these principles: Our highest priority is to satisfy the customer through early and CONTINUOUS DELIVERY of valuable software.”

Agile Manifesto (2001)
Continuous Delivery - How?

- Numerous tools can support Continuous Delivery
- Way of thinking about development process
Continuous Integration + Delivery

- Continuously integrate new features in working software code base, multiple times a day
- Automate (almost) everything
- Extensive tests secure existing features and help identifying integration problems as early as possible
- Immutable artifacts
- Every change is proven to be deployable at any time

ARW 2017, Jean-Christophe Garnier, CERN
Teamwork Process

- Design discussions
- Pair programming
- Code reviews
- Early user involvement to get feedback
- Source code quality must improve
- A task is done when it’s tested
Automated tests

- Unit Testing: a small unit of code in isolation
- Integration Testing: multiple components together
- User Acceptance Testing: the specifications are fulfilled
- Staging: validation in environment as close as possible to production

➢ Validation in operational environment requires less time if the other strategies are applied
➢ Test code must be maintained, documented, and must follow quality standards
➢ How is the test code validated?
System Validation - Source code
System Validation - Product

- Automation vs manual/human
  - Consistent
  - Repetitive
  - Fast

**Automated Analysis Results ±PCC.1**

- **PCC.1 Automated Analysis Results 2012-2013**
  - 398 PCC.1 tests on 350 60A-circuits
  - 10 consistency fails
  - 1 was “successful” but “tuning to be done”?
  - 8 cases due to “just” failing decay criteria
  - 1 anomaly accepted manually???
  - 97.5% of consistency!

Time comparison of Commissioning Test Analysis

ARW 2017, Jean-Christophe Garnier, CERN
Testing environment

- The smaller the scope of the test, the simpler the environment
- Unstable environment for acceptance tests
  - Shared hardware lab
  - Unexpected changes in the build tools
- Future is brighter
  - Moving to standard tools
  - Getting dedicated hardware in a testbed considered “semi-operational”
Quality Monitoring

- Public
- Build user confidence in our software
  - Developer using libraries
  - End user using GUI or services
Quality Monitoring

The only valid measurement of code quality: WTFs/minute

Good Code.

WTF

WTF

WTF

WTF

WTF is this shit.

Dude, WTF

BAD CODE.

WTF

WTF

WTF

WTF

(c) 2008 Focus Shift
Deployments

- Only if all tests are successful
- Automated deployment procedures
  - Rare maintenance
- Post deployment checks
  - Does the GUI/server start?
  - Are all the objects initialized properly?
  - Too few sanity checks at the moment
Outlook

● 3 successful years of continuous delivery
  ○ Removed environment complexity from work routine
  ○ Fearless deployments, even for newcomers
  ○ Very reactive on operational issues and changes of requirements

● Few legacy projects not integrated into it
● Automated post-deployment validation is missing
● Only scratched the surface today
Questions?
Perspectives - Back up slides
Return on Investment

- 3 successful years of continuous delivery
- Very reactive on operational issues and changes of requirements
- Full history of releases
- One click deployment/rollback
- Deployments take few seconds
- Relaxed developers and relaxed users
Continuous Integration vs Feature Branching

- Tools are changing, Team is changing, Mindsets are changing

- Still compatible with Continuous Delivery
- Less frequent merges
  - Waiting for more experience before judging

ARW 2017, Jean-Christophe Garnier, CERN
Deployment Service

- Remove developers from the operational system
- Deployments and rollbacks controlled by Operations Crews
- Tracking of all software versions
- Tracking of all dependencies between services and applications

➢ Part of it are in place, more focus needed
Outlook

● 3 successful years of continuous delivery
  ○ Removed environment complexity from work routine
  ○ Fearless deployments, even for newcomers

● Future developments will likely follow a different process:
  ○ What will be the impact on our productivity?
  ○ And on our software quality?

● Collaborative incentive to provide fully-featured deployment service

● Metrics collection about software defects and impact on availability/reliability