How long does it take your team to deploy a change that involves a single line of code? And do this on a reliable, repeatable basis?

Continuous Delivery: ..., Jez Humble and David Farley, 2011
Cycle Time is the key metric

- Time from deciding that you need to make a change to having it in production
- Years? Months? Weeks? Days?
- Some teams achieve a cycle time of hours or minutes

Continuous Delivery

- Agile - deliver early and continuously.
- First release is the just the beginning and contains the minimum amount of functionality
- Majority of cost occurs after first release in the form of maintenance, support, new features and defect fixes
Continuous Delivery

- Build - Deploy - Test - Release
- Releases become routine
  - Ideas rapidly turn into delivered code
  - Deliver more frequently with less stress
- Adds the last mile to Continuous Integration
  - turn integrated code into production software
  - more automation, done quickly without error
  - involves separate testing and Dev Ops teams

Continuous Delivery: ..., Jez Humble and David Farley, 2011

Continuous Delivery Principles

- Create a repeatable, reliable process for releasing software
- Automate almost everything
- Keep everything in version control
- If it hurts, do it more frequently, and bring the pain forward
- Build quality in
- Done means released.
- Everybody is responsible for the delivery process
- Continuous Improvement
Deployment Pipeline

- Every change creates a new instance of the pipeline.
- Every change is a release candidate.
- Done means released.
- No alpha - beta- ... - gold

Configuration Management

- Keep absolutely everything in version control
- Check in regularly to master
- Use meaningful commit messages
- Manage external libraries
- Manage components
- Manage application configuration
- Manage your environments
Testing Strategy

<table>
<thead>
<tr>
<th>Business Facing</th>
<th>MANUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOMATED</td>
<td>Showcases</td>
</tr>
<tr>
<td>Functional tests</td>
<td>Usability testing</td>
</tr>
<tr>
<td>Integration tests</td>
<td>Exploratory testing</td>
</tr>
<tr>
<td>System tests</td>
<td></td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Nonfunctional acceptance tests (capacity, security, …)</td>
</tr>
</tbody>
</table>

Continuous Integration

- Don't check in on a broken build
- Always run all commit tests locally before committing, or get your CI server to do it for you
- Wait for commit tests to pass before moving on
- Never go home on a broken build
- Always be prepared to revert to the previous version
- Time box fixing before reverting
- Don’t comment out failing tests
- Take responsibility for all breakages resulting from your changes
- Test-Driven Development
Continuous Improvement

• Optimize the whole process, not just the parts
  – Configuration Management
  – Automated Testing
  – Continuous Integration and Deployment
  – Data Management
  – Environment Management
  – Release Management
• Increase feedback and improve collaboration between development, testing, and dev ops teams

Other Metrics as Diagnostics

• Automated test coverage
• Codebase properties - complexity, duplication, coupling, …
• Number of defects
• Velocity
• Number of commits, builds, build failures per day
• Duration of builds, automated tests
Theory of Constraints

- Identify the limiting constraint in your process (bottleneck)
- Exploit the constraint (maximize the throughput for that part of the process - 100% utilization)
- Subordinate all other processes to the constraint (less emphasis on other parts of the process)
- Elevate the constraint (increase resources, investment)
- Find the next constraint and repeat

Continuous Delivery: ..., Jez Humble and David Farley, 2011