Building the Agile Enterprise: Combining Kanban and tooling to scale Agile beyond your team

Gil Irizarry
Karen Hannon

Constant Contact

March 2012
Agenda

• Background on Constant Contact

• Our Process

• Our Tooling
The Goal

• We want to release more features more frequently

• Why?

  • React quicker to changing market conditions
  • Get new features to users more quickly
  • Frequent releases are smaller releases: our customers often cannot absorb a large set of changes

• We also want teams to work more smoothly and continuously
The Challenge

- 13 software development teams
  - 5 development and testing environments
  - 2 production environments
- A code base that was first created around 2004 and built up over time
What do we have?

A mess of interdependencies!
So, how do we address this?

- With Scrum of Scrums!

- Um, wait…
Scrum of Scrums (of Scrums (of Scrums...))

- How to integrate teams in Scrum?
  - Hold a Scrum of Scrums
- What happens if you have too many teams?
  - Hold a Scrum of Scrums of Scrums, and so on...
  - Add a MetaScrum or two
Our Process

- Kanban at the team level
- Release Management at the org level
- Manage the intersection of the two
5 Core Properties of Kanban

• Visualize the workflow
  • States in the team board are a reflection of how a team completes its work

• Limit work-in-progress (WIP)

• Manage Flow
  • Implied that flow should be continuous

• Make Process Policies Explicit

• Improve Collaboratively
Release Management

• Prioritized project list
• All At Once Planning
• Classes of Service

• (Near) Continuous integration and (mostly) automated regression testing
• Dependency and deliverable review
• Release train
• Product Management, Engineering and Operations leads meet regularly to prioritize all development projects
• This becomes the organizational backlog, which drives each team’s backlog
All At Once Planning

- Prior to the start of a release, teams use the prioritized project list to plan their upcoming work
- Planning involves the identification of deliverables and dependencies
- Dependencies are discussed with dependent teams
- A meeting is held in which all development and operations teams present their dependencies to each other
All At Once Planning

• At the end of the meeting, each team has their planned deliverables and incoming dependencies

• If they haven’t already, they determine their capacity and, based on the priorities, commit to a set of work

• This means that a team may have capacity to do work, but may not get to it in a release if that work pushes the operations team beyond its capacity

• Once a release starts, we will have each team’s set of commitments
Dependency and deliverable review

• Project leads review the deliverables and dependencies to which they have committed and say if they are on track or not

• We found the teams could still miss a deliverable even if they had no impediments. Deliverable tracking provides a better view of the state of the release
• Releases are iterative but development is continuous
Release Train

• If a deliverable misses a release (the train), it simply waits to capture the next one

• We don’t penalize a team if a deliverable is not done at the end of a release and misses the release train

• Teams plan and test continuously

• It’s OK if a team pulls functionality from a release within a release cycle

• It’s OK if a team starts work for the next release in the current release
Release Train

- To make this work:
  - Releases must be short
  - Must have tooling for automated builds and deployments
  - Other areas of the organization must be flexible enough to react to last-minute product changes
So, How Do We Make This Happen?
# We Updated Our Tooling

<table>
<thead>
<tr>
<th></th>
<th>The Past</th>
<th>Today</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Control</td>
<td>ClearCase</td>
<td>GIT</td>
</tr>
<tr>
<td>Continuous Integration</td>
<td>None</td>
<td>Jenkins</td>
</tr>
<tr>
<td>Builds</td>
<td>Ant</td>
<td>Maven</td>
</tr>
<tr>
<td>Environments</td>
<td>Desktop PCs</td>
<td>Macs/Vagrant</td>
</tr>
<tr>
<td>SQL</td>
<td>Run manually</td>
<td>Liquibase</td>
</tr>
</tbody>
</table>
We Updated Our Architecture and Infrastructure

• Transitioning from a monolithic codebase to a service based architecture

• And from a single DB2 database to application specific MySQL databases and Cassandra

• Moved from Websphere to Jboss

• Package Code, Configuration, SQL, and Tests together

• Automated our deployments with Puppet
Increased Emphasis on Testing

- Monitoring code coverage
- Advancing our automated test tooling; JUnit, Selenium and Rspec
- Unit tests run at build time
- Automated Smoke and Integration tests run in our CI environment
- Code not advanced if tests fail
- Daily emails to entire team on testing status
Evolving Branching Strategy

- Branch per application/team
- Initial development in developer’s local branch
- Team integration in Development branch
- Multi-Team integration in the Integration branch
- On the train – promoted to Release branch
- A Master branch for current production
Development workflow
Continuous Delivery

Brings together:

- Continuous integration
- Automated testing
- Automated deployment

Resulting in:

- Stable test environments
- Increased developer productivity
- Higher quality releases
- Smaller, more frequent releases
Takeaways

• Decouple team planning from releases

• Make releases small and frequent

• Code, architecture and infrastructure must work together to make small, frequent releases possible

• Plan and prioritize continuously

• Automate Everything! (Building, Testing, and Deploying)