Continuous Delivery of Software

Reducing risks with systems, feedback and flow

SEPG North America 2013
Joanne Molesky
October 3, 2013
Purpose

• Challenge traditional concepts for software delivery
Agenda

• What is Continuous Delivery?
• Continuous Delivery principles and concepts
• Implications for organization process development
• Conclusion
• Questions
What is Continuous Delivery?
What is Continuous Delivery?

An approach to delivering software that reduces the cost, time, and risk of delivering incremental changes to users.

Fast, automated feedback on the production readiness of your applications every time there is a change whether code, infrastructure, configuration or database.

Jez Humble and David Farley
Continuous Delivery Concepts

Enables:

• Software that is always production ready
• Minimized lead time from idea to live
• Decisions based on real data and empirical evidence
CD is not:
Continuous Release
Crash Course on Evolution of Software Development
A ‘classic’ software release

- **Design**
- **Build**
- **Test**
- **Release**

This should work. Here is is your documentation.

I think this is the way it should go together.

Well, this isn’t right!

No more time or money left, put it in, warts and all.
A ‘classic’ software release

- Defect Introduced
- Stuff built on top
- Feature incorrectly implemented

Deployment Issues!
Bloat
Defects!
Business has Changed
Agile Software Delivery
Activities typically pulled forward:

- Testing of completed stories by QAs
- Review & sign off stories by stakeholders
- Feedback incorporated into following iterations
- Release testing
- Release to production
We built the basics. How does it look? What do we do next?

This is what we think it will look like and how we want to start.

We built the next part. How does it look? What do we do next?

You want something new? What shall we drop?

This is what is good to go

Deployment Issues!

Support Issues
Continuous Delivery
Deployment Pipeline - Treat Everything as Code

- Operating environment update
- PAF update change request
- Story / defect iteration
- Java initiation
- Continuous integration
- Acceptance testing
- Performance testing
- ETL regression
- Dry run
- Production
- Data patching
- Control-M

ThoughtWorks
Continuous Delivery: small incremental change

Series of tested small commits

Unsuccessful changes aren’t allowed in the system
Every successful change is a release candidate
Continuous Delivery: small incremental change

Releases on demand
The benefit: Releasing becomes a business decision
Repeatable
Reliable
Releases
Deployment Pipeline

Deployment Pipeline = Greater Control

- Complete traceability
  - No emergency
  - Fast recovery
- Test ideas
Repeatable Reliable Releases

Automation

• Less human error
• Faster service delivery
• Creates opportunities to practice
• Improves transparency
• Enables monitoring and measurement
Repeatable Reliable Releases

• Leverage virtual machines and IAAS
• Treat servers like cattle - NOT pets
• Continuous integration of code – eventually no branching
• All changes managed in source control
• Creation of environments
• Deployments to all environments
• Automated testing
Flow
Functional Silos Create Waste
Flow: Small Incremental Change

Delivering smaller incremental change = Time and money saved, better product

- Delivers business value faster
- Easier to locate mistakes
- Easier to fix
- Build only what you need
Flow - Deliver Value

Traditional Approach Untested/
Unreleased changes = risk

Measurable Value = 0

Code

Time

Working Software
Flow - Deliver Value

Tested released changes = less risk
Released changes = measurable value

ThoughtWorks
Flow - Easier to Fix

uh oh
problem happens

???*!!##^%**#??!!

time to figure out cause of problem

wheeew
problem solved

adapted from John Allspaw
Flow - Build Only What You Need

Standish Group: how often features are used

- Never 45%
- Rarely 19%
- Sometimes 16%
- Often 13%
- Always 7%

Stop when you have enough
Feedback
Continuous Delivery Principle: Early Feedback

Early frequent feedback = Better product

• Fewer Defects
• Build the right thing
Early Feedback

- Support programming: functional tests, acceptance tests
- Critique the product: showcases, exploratory testing, usability testing
- Business
- Technology: unit tests, component tests, system tests
- Performance tests, security tests
Collaboration
Collaboration = Better Outcomes

- Less waste
- Improved understanding
- Better decision making
- Shares responsibility
- Improves transparency
Typical IT Organisation

Business

Engineering

Project C
Project A
Project B

Operations

DBAs
Service Desk
Infrastructure team
Think Product and Services

Products / Services

Operations

- IaaS
- Ops management
- Service desk
Solution Development

- Get rid of siloed functional teams
- Focus on people and outcomes first, then process
- Reduce bottle necks and waste
  - Encourage self service
  - Encourage collaboration
- Increase transparency and visibility
  - Allow for experiments
Change Management Process

- One size does not fit all
- Decisions s/b made by those with knowledge and skin in the game
- Focus on outcomes, not activity
- Store evidence in the deployment pipeline
- Allow exceptions and improvement in the process
Project Management

- Think **products**, not projects
  - Manage **Portfolios**
- Still need product development management
- Focus transparency and visibility of what is done
- Use empirical data to make decisions
Budgets and Financial Management

• Don’t focus on annual budgets
  • You still need to plan
• Manage product costs not project costs
• Rethink CapEx/OpEx buckets
  • Allow for experimentation
Conclusion
References

➤ Continuous Delivery
   • Jez Humble, David Farley
   • Addison-Wesley, 2011

➤ Lean Enterprise
   • Jez Humble, Joanne Molesky, Barry O’Reilly
   • To be Released O’Reilly Books, 2014

❖ Why Enterprises Must Adopt Devops to Enable Continuous Delivery
   • Jez Humble, Joanne Molesky
   • Cutter IT Journal, 24:08 pp 6-12, August 2011

◆ http://continuousdelivery.com/

◆ http://www.thoughtworks.com/continuous-delivery/
Q&A