Gene Gotimer
Experiences Bringing Continuous Delivery to a DoD Project
About Coveros

- Coveros builds security-critical applications using agile methods.

- Coveros Services
  - Agile transformations
  - Agile development and testing
  - DevOps and continuous integration
  - Application security analysis

- Agile & Security training

- Government qualifications
  - DCAA approved rates and accounting
  - TS facility clearance
Select Clients
The Project

- COTS product integration for DoD
  - custom Python glue
  - and Java, PHP, Perl
- Releases every 6 months or so
  - Freeze 2-4 weeks in advance
  - Deploy Friday evening to Sunday afternoon
  - Repair broken functionality Monday and Tuesday (and on)
- Barely starting Agile
  - Daily Stand-ups
    - (really daily status calls)
  - 2-week Sprints
  - Good, pruned backlog
  - No automated testing
  - No unit tests
  - No continuous integration
The Delivery Team

- Development – Local
  - 2 Developers
  - 1 Business Analyst
  - 1 Project Manager

- DISA PMO
  - 1 Program Manager
  - 1 Chief Engineer
  - 1 Program Director
  - 1 Systems Engineer

- Test and Integration – Remote
  - 4-6 Testers
  - 4-6 Integrators
    - including security experts
  - 1 Information Assurance

- Off-team
  - Systems Administrators
    - hardware and software
The Problem

-\_(ツ)_/-

“It works on my machine!”
- Every developer, at some point

= HIGH RISK DEPL OYS
DevOps is...

“How long would it take your organization to deploy a change that involves just one single line of code? Do you do this on a repeatable, reliable basis?”

- Mary and Tom Poppendieck

Implementing Lean Software Development: From Concept to Cash
DevOps is...

“The goal of DevOps is not just to increase the rate of change, but to successfully deploy features into production without causing chaos and disrupting other services, while quickly detecting and correcting incidents when they occur.”

- Gene Kim

Top 11 Things You Need to Know About DevOps
Continuous Delivery

• Make releasing a business decision, not a technical decision

• High-confidence releases
  • Small releases
  • Fully tested
  • No expectation of problems

• Hotfix releases
  • Possible, no more than moderate risk and moderate coordination
The Approach

• Started with things that were in our control
  • Dev and Test environments
  • Development process

• Make changes behind the scenes
  • Free/open source tools
  • Easy to integrate into our CI system
  • Small changes

• Disclose the changes when there was a win
  • Highlight ease of use
  • Use as justification for higher environments
The Journey

1. Continuous Integration
2. Functional Testing
3. Automated Deploys
4. Security Testing
5. Performance
6. Culture Clash

4½ Years
September 2009 – March 2014
1. Continuous Integration

- Trouble explaining “integration”
  - between two or more developers
  - not between systems
- Set up SecureCI one afternoon
- Explained the advantages later
- Wired to the ALM tool we had
  - Jenkins (Hudson at the time)
  - Nexus
  - SonarQube (Sonar at the time)
- Automated builds
  - Ant, Maven
  - PMD, FindBugs, Checkstyle
  - Cobertura
  - Later added Python tools
1. Continuous Integration

- This gave us a strong basis for CD later, although we didn’t know it at the time.

**Lessons Learned:**

Continuous integration is valuable, but outside the dev team it isn’t obvious. The biggest advantage to open-source tools is often acquisition time, not acquisition cost.
2. Functional Testing

• Functional testing was done manually
  • from a script written in Microsoft Word

• We waited a year before staging a coup
  • we didn’t want to encroach on their domain

• Demo of Selenium
  • demonstrated record-and-playback through the Selenium IDE
  • we recorded the first set of tests
  • then turned it back over to the test team
2. Functional Testing

• They argued later that automated testing was ineffective
  • the automated script (singular) only worked one time
  • needed to be re-recorded when any changes got made to the app

Lesson Learned:
Automated testing isn’t just about replacing manual tests with an automated test framework. It requires a different way of thinking.
2. Functional Testing

• We took it back
• Rewrote existing tests in Java
• Showed our business analyst how to clone-and-mutate the Java tests
• Started with JUnit, but went to TestNG
  • better tagging and parameterization
  • pre-test run initialization
2. Functional Testing

• Development team had more confidence in releases
• Also began testing user roles
  • Security testing = what can this type of user NOT do

Lesson Learned:

Should have focused on demonstrating that there were fewer escaped defects.
It was hard to point to a clear benefit.
The Book

- Project Manager came across the book in a book store
- Everything made so much sense
- Logical extension of what we were trying to do
- Addressed a lot of the issues we were running into
- No money or time for an effort, so we adopted it as our long-term goal
3. Automated Deploys

• Started with automating a Drupal web server install
  • new system, not yet in production
  • database server was easy, so we skipped it for now
• Then automated the manual COTS install
• Then started reverse engineering the broken COTS installer
3. Automated Deploys

• Down the road, realized we could automate everything
  • Doesn’t just reduce risk, also speeds up the process

Lessons Learned:
Automate everything—even the easy stuff.
When it is easier to install, you’ll stumble across more reasons to install it.
Go from Why? to Why not?
3. Automated Deploys

- No Puppet Enterprise Server
  - just manually ran puppet apply from the command line
  - every system (DB, Web server, SVN server, ALM tool) used the same puppet apply command

- Vagrant would have been helpful for local deploys
  - Just hadn’t heard of it
4. Security Testing

• Decided we needed at least some security in dev
  • System hardening
  • Web application scanning
• We knew it couldn’t replace the “official” testing
  • plus, we didn’t want to encroach on their domain

• Noticed extra processes running
• Dev system in cloud with default password
• Tested Security Blanket
  • just purchased by Raytheon
  • couldn’t get it purchased
4. Security Testing

• Knew we had some good base for security
  • CI, static analysis, user role testing
• Wanted a security scanner
  • at the time, none worked with client certificates out of the box
• Found w3af
  • Python
  • customizable
  • client certificate support was there, but not exposed
  • handed it over to the “security experts” on the integration team
4. Security Testing

Found 0 vulnerabilities!
4. Security Testing

- Never got past the login screen
- Never read the output or log
- So we took it back
  - Eventually had problems getting customized w3af to work properly
  - Switched to OWASP ZAP, run manually
- Security team focused on STIG and SELinux
  - that was their expertise anyway
4. Security Testing

• Lost a lot of faith in us when we were hacked
• Information Assurance isn’t the same as Security

Lesson Learned:
Protect every system, everywhere.
Many hacks are just for the system, not the data.
4+. Security Testing

- Over a few days, implemented OpenSCAP in Jenkins for STIG
  - immediately found issues
  - started adding Puppet manifests for remediation
- Started using Nikto2 for web server scanning
  - immediately found issues
- Started running weekly scans of dev and test using OpenVAS
  - no immediate issues, but started seeing package security updates before they became IAVMs
- Discovered SELinux was in permissive mode
  - had never been in enforcing
4+. Security Testing

• Easier audits
• Proactive security upgrades
• Much better relationship with the data center

Lesson Learned:
Benefits of security testing go beyond increased security.
5. Performance

• Applying STIG to database server
  • seemed like it was getting slower
• Used JMeter to get baseline
• Took rough breakdown of most common queries
• Repeated as a 15-minute test
• Monitored trend

• Added similar testing to functional tests, another 15 mins
• Also, number of functional tests was growing slowly
• Watched functional test elapsed time as rough guide
5. Performance

• Watching trends can be very worthwhile
• Some testing can be almost as valuable as full testing

Lesson Learned:
A baseline can be a great safety net.
6. Culture Clash

- Continuous Delivery was being openly discussed
  - PMO had just started thinking of it as a clear plan
  - Kept asking when “continuous delivery” would be delivered, and how it would be packaged

- Test and Integration started complaining
  - 3½ of us were pushing the 12+ of them too hard
  - moving too fast
  - not a risk or control complaint, merely effort

- People on test and integration team started leaving
  - including “Burt”
6. Culture Clash

• Benefits were growing clear
• Effort was minimal
• No active resistance

Lesson Learned:
Do not underestimate cultural inertia.
Some will not or cannot ever make the mental shift.
The Aftermath

- Test and Integration decided not to renew their contract
  - all remaining personnel ended project with a month
- Security issue found the following week
  - deployed 3 days later
- Went back to 2-week deploy cycles, sometimes faster

- Left 3 people on development team
  - One went back to take over for the test and integration team as hands-on-keyboard
  - BA left project and another came in ½ time for testing
- Dropped into maintenance mode
The Delivery Team

- **Development – Local**
  - 1 Developer
  - 1 Release Manager
  - ½ Tester

- **DISA PMO**
  - 1 Program Manager
  - 1 Chief Engineer
  - 1 Program Director
  - 1 Systems Engineer

- **Test and Integration – Remote**
  - 1 Information Assurance

- **Off-team**
  - Systems Administrators
    - hardware and software
The Project

• Barely Agile
  • Maintenance only
  • Kanban-ish
    • tracking work in progress
  • Daily Stand-ups
    • (really daily status calls)
  • 2-week Sprints

• Releases prepared every 2 weeks
  • Soft freeze Thursday for Friday release
  • Deploy Friday evening
  • 100% working functionality Friday evening
  • Non-event

• Puppet took the configuration parameters
  • from 200+ untracked values
  • to ~30 Hiera-controlled values

Biggest coordination issue:
72 hours for user messaging

Biggest time consumer:
3-6 hours for VM clones
My Advice

Lessons Learned:

DevOps and Continuous Delivery are not a goal. Do not set out to do DevOps or CD. Remove road blocks and bottlenecks. Fix quality issues. Be more responsive to change. Adopt change incrementally. As you build a repeatable, reliable process for delivering software, CD will “magically” appear.
My Advice

• Read Continuous Delivery and The Phoenix Project
Missed Opportunities

• Automated deploys
  • more valuable than just reducing risk
• Vagrant
• Some security scanning earlier
  • do not just assume someone else is doing it
• Some performance testing earlier
  • some is a lot better than none
  • maybe almost as good as a lot

• We relied on client-side certificates for authentication
  • EJBCA should have been set up immediately
• Upgrades are a huge time sink
  • components, libraries, applications, system software
  • add tools to track it as early as possible
The Tool Chain

• Jenkins
• Puppet (no Puppet Enterprise)
• Testing
  • TestNG for Java unit tests
  • Nose for Python unit tests
  • Mockito/Mockito for Python
• Static Analysis - Java
  • PMD
  • FindBugs
  • Checkstyle
  • Cobertura
  • SonarQube
• Static Analysis - Python
  • Pylint
  • coverage.py
• JMeter
  • for some representative performance tests
• Security
  • OpenSCAP (every deploy, minutes)
  • OpenVAS (every weekend, hours)
    • included Nikto2
    • used Kali Linux
  • OWASP Dependency Check (on-demand, many minutes)
  • OWASP Zed Attack Proxy (on-demand, few days)
  • Full role-based Selenium test coverage (every deploy, overnight)
    • 10k+ Selenium tests via TestNG
Questions?

Gene Gotimer
gene.gotimer@coveros.com
@CoverosGene