Transitioning Your Software Process To Agile

Jeffery Payne
Chief Executive Officer
Coveros, Inc.
jeff.payne@coveros.com
www.coveros.com

About Coveros

- Coveros helps organizations accelerate the delivery of secure, reliable software

- Our services:
  - Agile software development
  - Application security
  - Software quality assurance & test automation
  - Software process improvement

- We focus on industries with significant regulation or compliance requirements
  - Financial services & Insurance
  - US Defense & Intelligence
  - Homeland Security & Critical Infrastructure
  - Healthcare & Medical

Pop Quiz: Agile Development Means …

- No documentation. We don't need to write anything down!
- No process. We can do it any way we want!
- No overtime. We can go home at 5!
- No management. We decide when to deliver!
- No testers. Who needs them anyway!

What Agile Actually Is

- An approach to software development that recognizes that building software is much more a design process than a construction process
  - Adaptive over Predictive
  - People over Process
  - Visibility into the Process!!!!!

- Agile Methodologies
  - Extreme Programming
  - SCRUM
  - Lean Development
  - Crystal
  - Agile RUP

Common Agile Practices

- Management Practices
  - Daily standups
  - Retrospectives
  - Release planning
  - Iteration planning
  - Requirements envisioning
  - Visible progress reporting
  - Risk management
- Development Practices
  - Iterative development
  - Test driven development
  - Continuous integration
  - User acceptance testing
  - Refactoring
  - Rapid prototyping
  - Pair programming

Common Mistakes in Transitioning to Agile

Mistake #1 – Trying to do too much too fast

- Change is HARD for most people
- Adoption of agile methods often fails when organizations try to radically change too much too quickly
- Best practice:
  - Make changes gradually to assure improvement and organizational buy-in
  - Choose first those changes that are 1) easy and 2) foundational
  - Pilot changes to demonstrate success

Mistake #2 – Hiring the wrong help

- Progress is made in the trenches and is context specific
- Beware of the consultant who:
  - hasn't built a real software product in years
  - thinks coaching & classroom training is all it takes
- Recommendations must be practical to gain acceptance and stick
- Best practices
  - Hire a professional consultant who has recent real-world experience

Mistake #3 – Trying to Become Agile instead of agile

- Business value is the goal, not becoming Agile
- You don’t need to adopt a particular Agile Methodology to gain the benefits of agility
- Best Practices
  - Start from where you are (Duh!)
  - Leverage agile methods that have the largest impact on business value (correct the largest weaknesses quickly if possible)

Mistake #4 – Forgetting About Management Practices

- Technologists often love to focus on development activities and tools
- Often our biggest problems in software are due to lapses in communication or project execution
- Best practices
  - Move forward with both management and development practices together for biggest impact

Mistake #5 – Forgetting About Testing

- Adaptive processes need feedback to be effective!
- Testing provides valuable feedback on the actual state of the project not only in terms of quality but schedule & cost as well
- Early lifecycle testing & test automation can also substantially accelerate your feedback loop
- Best practices
  - Make sure improvements are made in testing right along with development
  - Leverage Continuous Integration to automate your testing process

Mistake #6 – Forgetting to Demonstrate ROI Early

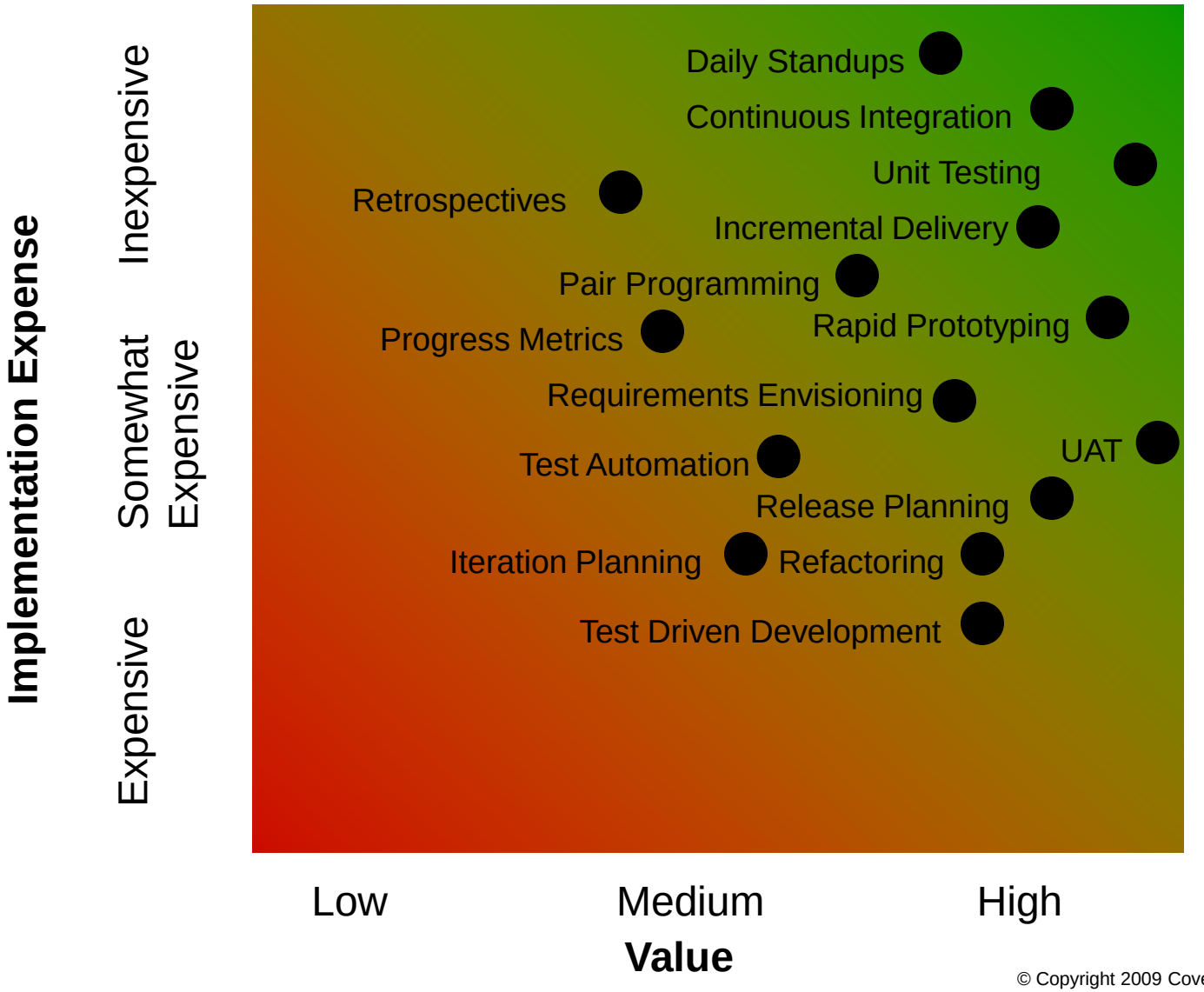
- Improvement initiatives often die due to the team's inability to demonstrate ROI for $$$ spent to improve
- Make sure you show *short-term* ROI!
- Best practices
  - Capture the value of your improvements in terms of:
    - Productivity (aka velocity)
    - Quality (decreases in defects and rework)
    - Time-to-market (hitting delivery dates)

Mistake #7 – Forgetting to Include Users

- A key piece of agile development is early, frequent feedback from users, customers, stakeholders
- Make sure you include these constituents in the process
- Make sure business analysts are involved as well
- Best practices
  - Proxy for busy business executives
  - Users involved in user acceptance testing
  - Prototyping to make requirements ‘real’

High Value Agile Practices

Return on Investment from Agile Methods



Incremental Delivery

- Development of complete, fully tested features in short (2 – 4 week) increments
- Fundamental to supporting any type of agile development process
  - Allows for rapid quality feedback
  - Allows for rapid user feedback
- Drives ability to adapt, gain rapid feedback, and provide ultimate flexibility

Daily Standups

- Brief (10 – 15 minute) daily team meetings
- Assures continual communication on the team
- Drives accountability (peer-pressure, visibility)
- Demonstrates day-to-day progress to all team members and stakeholders

Continuous Integration

- Continual build and test of software
- Typically supported by automation and integrated with source code control, defect tracking, measurement tools
- A building block for test automation
- Drives automation of iterative development & feedback loop

Rapid Prototyping

- Development of a high level prototype (often of the user interface) for user feedback
- A great way to make requirements “visual” for stakeholders and customers
- Often developed as a throw away but frequently is built upon anyway (watch this!)

Requirements Envisioning

- Upfront requirements analysis that results in a set of related artifacts for project management
  - Release plan – overall plan for product development
  - First iteration plan – definition of what features will be built first
  - User stories & backlog – Requirements and priorities
  - Rapid prototype – Visual example of the system to be developed

- Overall program plan
  - Incorporates the above plus staffing, costs, risks

Test Driven Development

- Tests developed as part of building code
- Useful for specifying features not fully understood
- Assures unit testing is performed
- Often uses code coverage as a criteria for test completeness
- Tests often automated and rerun during incremental builds

Ways to Get Started

Which Practices to Consider Doing First?

	Management Practices	Development Practices
Thereafter	Risk management	Application Security
	Compliance	Rapid Prototyping
	Retrospectives	Test Automation
Next	Project metrics	Use cases
	Progress dashboard	UI wire frames
	Requirements envisioning	Unit testing
First	Release & Iteration planning	User stories & backlog
	Daily standups	User acceptance testing
	Iterative development	Continuous integration

What development methodology are you following today?

- Waterfall – Straight line development of requirements, architecture / design, code, tests
  - New development
  - Supporting legacy systems
- RUP – Rational Unified Process that includes incremental development, use cases, UML for design, IBM/Rational tools for dev and test
- Ad-hoc – Cowboy programming

Moving from Waterfall

- Waterfall is not incremental. Must tackle move to increment development approach or agile is not possible
- Begin with a requirements envision to prioritize requirements and begin doing ‘mini waterfalls’ in 3 month increments
  - Reduce to 1 month increments over time & add automation
- Legacy development you don’t control
  - Focus on leveraging agile management instead of trying to convince the dev team to change
  - Use management techniques to gradually shift them to automation & better testing

Moving from RUP

- IBM is putting a significant amount of resources and though leadership around Agile RUP.
- Agile RUP is a lightweight RUP process that more closely aligns with the agile philosophy
- Suggest anyone doing RUP start by moving toward agile RUP

Moving from Ad-hoc

- Use continuous integration to begin adding structure and feedback to a chaotic process
- Begin with simple hurdles for checking in code (code must build) but collect information on testing that is performed
- Slowly ‘raise the bar’ around regression testing and unit testing to not allow code to be check in unless it is tested and then that the tests all pass
- Make build failures visible to the entire team

Questions?

Thank You