Lean / Agile Programming in a Mainframe World

by: Zamir Gonzalez
z Tools and Transformation Team
Agenda

- **Lean and Agile what’s behind the hype**
- Lean & Agile Defined
- Before: Waterfall
- After: Agile
- Transformation:
  - How we made it work
  - Lessons learned
- Tooling that made it possible
  - Rational Team Concert
If it ain’t broke, don’t fix it
**Geographic Barriers**: poor communication, language, culture and time differences, process gaps resulting in errors and rework, high degree of friction across teams

**Organizational Barriers**: lack of meaningful stakeholder input, poor line of business oversight, weak project governance, missed opportunities to leverage domain expertise

**Infrastructure Barriers**: incompatible tools and repositories, unreliable access to common artifacts, lengthy project and team on-boarding, brittle and inflexible tooling integrations
Collaborate in Context: enable team transparency, build team cohesion, automate hand-offs

Right-size governance: automate workflows through dynamic processes, automate data collection, real time reporting and alerts

Day one productivity: dynamic provisioning of projects and teams, real-time release/iteration planning and workload balancing, unify teams
Lean & Agile – Process Perspective

- Users seldom know exactly what they want
- Many details that can only be discovered well into implementation
- We can master only so much complexity
- External forces lead to changes in requirements
Agenda

- Lean and Agile what’s behind the hype
  - **Lean & Agile Defined**
  - Before: Waterfall
  - After: Agile
  - Transformation:
    - How we made it work
    - Lessons learned
  - Tooling that made it possible
    - Rational Team Concert

---

7
Lean Defined

“Think big, act small, fail fast; learn rapidly”

Lean Principles:

**Eliminate waste**: Everything not adding value to the customer is considered to be waste. This includes: unnecessary code and functionality, delay in the software development process, unclear requirements, bureaucracy, slow internal communication.

**Amplify learning**: scrums, short sprints allowing testing and feedback to come quickly, reflections session at end of sprint for group lessons learned

**Decide as late as possible**: use options-based approach for delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle – the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

**Deliver as fast as possible**: just in time production ideology utilizing scrums and sprints

**Empower the team** -- developers should be given access to customer; the team leader should provide support and help in difficult situations, as well as make sure that skepticism does not ruin the team’s spirit.

**Build Integrity in** -- The complete and automated building process should be accompanied by a complete and automated suite of developer and customer tests, having the same versioning, synchronization and semantics as the current state of the System. At the end the integrity should be verified with thorough testing, thus ensuring the System does what the customer expects it to.

**See the whole**
Lean: Eliminate waste

Internal Paperwork  Backlog = Delivery Delay  Wait time = Lost $$$  Red Tape & Change Control

Complexity  Defects  Extra Features Drive Cost Exponentially
“Uses continuous stakeholder feedback to deliver high-quality, consumable code through user stories and a series of short, iterations.”

Core principles
“fits just right” process
continuous testing and validation
consistent team collaboration
rapid response to change
ongoing customer involvement
Frequent delivery of working software
At Start of Iteration
- Development Process to be used (One Page)
- Current Candidate List

At Start of Coding
- List of Prioritized Selected Use Cases / Features to be delivered this iteration
- Latest Architecture / Model Design Docs (not maintained: Frozen point in time. NOT auditable)

At End of Iteration
- Demo
- Delivered Code
- Test Cases
- Reflection / Status
  - List of Use Cases / Features actually delivered (complete and tested)
  - Use Cases / Features not delivered (input to reflection)
  - Revised Development Process for next iteration

Product Backlog:
- A list of all desired work on the project
- Ideally expressed such that each item has value to the stakeholders
- Prioritized by the Product Owner
- Reprioritized at the start of each Sprint
The Sprint

Sprint Planning Meeting

Sprint Prioritization
- Analyze & Evaluate product backlog
- Select Sprint Goal

Sprint Planning
- Decide how to achieve sprint goal
- Create sprint backlog (tasks) from product backlog (stories)
- Estimate tasks in hours

Sprint Goal
Sprint Backlog
Agile Elements

- User Story
- Epic
- Story Points
- Planning Poker
Roles:

Allows the team to think of the product in terms of solving needs of real people
Identifies a type of user engaged in reaching some goals with your product
Provides the team insight into what the person is engaged in doing – although not necessarily

Goals:

Needs to represent the user’s goal
Should not be the technical solution
Should not be focused on advantages to the programmers developing the product

Business Value:

Allows the benefits to the customer to be apparent
Provides insight so stories can be intelligently prioritized
They are relative to the points of your other User Stories
    There is no sense of time in the measure

Establish 1 as a very simple effort, 5 as average, etc.

The team collaborates to size each user story

Accuracy is improved with history
A way to help teams to estimate User Story Points

• Each player has a Planning Poker deck

• Each deck consists of 13 cards: ?, 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100, and inf

Every User Story is estimated

• Points should include all the work to complete the story within the sprint
  • No matter what your role is, your card should take into account all required work

• Players place their cards simultaneously

• Players who play a higher or lower value explains why, and whole team play
Agenda

- Lean and Agile what’s behind the hype
- Lean & Agile Defined
- **Before: Waterfall**
- After: Agile
- Transformation:
  - How we made it work
  - Lessons learned
- Tooling that made it possible
  - Rational Team Concert
Waterfall is very much like a relay race with one fixed phases and sub teams waiting for traditional handoffs before they can begin.
IPD: Integrated Process Development
PCR: Process Change Request
DCP: Decision Checkpoint
zOS Process: Development

1. Brand: Line Item
   Emerging from Brand, our customer requirement has become one or more candidate Line Items for the upcoming release.

2. Design: Concept SLD
   Emerging from the initial stage of Design, each line item has an approved System Level Design (SLD) for Concept Phase.

3. Design II: Plan HLD
   Emerging next from Design, each SLD has a corresponding High Level Design (HLD) for the Plan Phase.

4. Development: Dcut Code
   Emerging from Development, each HLD has implemented code that has been completely Unit Tested.
zOS Process: Test & GA

1. **Function Test: FCT Exit Code**
   Emerging from Function Test, 100% of the function test cases have been executed, 98% are successful, and there are no high severity defects remaining.

2. **System Test: SVT’d Code**
   Emerging from System Test, the overall solutions & themes have been tested on real hardware.

3. **Still More Test: GA’d Function!**
   After the code has been unit tested, function tested, system tested, integration tested, performance tested, early-support-customer-tested, it becomes Generally Available with the release! But the life-cycle is not done yet...
zOS Process: Service Stream

Service: APAR
Emerging from Level 2 Service working with the customer, a software bug has been diagnosed and an APAR has been opened.

Service II: PTF
Emerging from Level 3 Service (Development & Function Test), the bug has been fixed in the form of a PTF, installable by any affected customer.

Customer: A follow-on request
The customer enjoys the new functionality, but always wants more. The process repeats...
Agenda

- Lean and Agile what’s behind the hype
- Lean & Agile Defined
- Before: Waterfall
- **After: Agile**
  - Transformation
    - How we made it work
    - Lessons learned
  - Tooling that made it possible
    - Rational Team Concert
zOS Agile Teams

Some development teams are fully lean and agile
- Leveraging strategic tools for team workflow
- Working for greatest impact, within a short sprint
- Delivering code rapidly
- Consumable code every two - six weeks
IPD: Integrated Process Development
PCR: Process Change Request
Benefits

- Improved understanding by entire team
- Increased team communication (local & remote)
- Improved usability due to stakeholder feedback
- Earlier removal of defects or design flaws
- Quick fix turn around
- Improved test efficiency (no lulls, less overlap)
- Better task tracking
- More effective/timely publication reviews

Earlier removal of defects as a result of parallel efforts:
  - solution provided to testers within days
    - in waterfall, would have been found months later, with formal defect process taking weeks

Increased team communication as a result of planning poker and smaller user stories
  - improved understanding by entire team
  - allows for better planning
  - increases morale
Scrum

- Increased and improved communications
- Continually re-focuses work activities and priorities

Parallel Development and Function Test

- FVT test cases available and executed during Unit Test
- Shift-left of discovery of errors, closer to when code was developed
  - Not having to write up all defects
- Function testers influencing code development
  - Actively participating in design and code inspections
  - Pointing out defects before code is written

User Stories & Tasks

- Rebalancing tasks among team enhanced
- Apportioning time over the effort -- spreading out the overtime
- Pieces of a line item can be deferred
• Predictable pace
  • Overtime peaks are distributed more evenly
  • Far easier to plan/estimate smaller bits of work
• Design churn minimized
• Real “consumables” available earlier
  • Earlier feedback from System Test, Level 2, exploiters, etc.
Voorstellen:
-wie ben je
-Wat doe je
-Waar zit je
-Wat hoop je / verwacht je

Huisregels:
-Interactie!
-Open minded
-Pauze om 19.30u
-Toetsing o.b.v. attitude en inzicht
**Burn Down charts** give visibility into a project's progress. They show the progress made against predictions, and open the door to discussions about how best to proceed, including the difficult discussions about whether to cut scope or extend the schedule.

**Velocity**: How much work you did in your previous iteration. It's usually measured in Story Points.

**Tech Debt**: The underlying cause of the inability to develop new features due to a defect burden

**Working Software** Remember one of the major guiding principles of Agile is: Working software Over comprehensive documentation. Attributes of working software include: (but not limited to)
Tested, stable, concrete (no sev1 or sev2), demoable to customers
Taken together, these iteration metrics and their trend over time provide an ongoing indicator of the team's real progress.
Here are some of the more common Agile metrics

**Velocity Chart:** A velocity chart shows the sum of estimates of the work delivered across all iterations. Typically velocity will stabilize through the life of a project unless the project team make-up varies widely or the length of the iteration changes. As such, velocity can be used for future planning purposes.

**Iteration Burndown:** Task progress provides a very telling measure of overall iteration progress and has the potential (though it often does not) to remain at a constant rate throughout an iteration. The Burndown Chart shows a trended view of task progress and is the most common measure of iteration progress.

**Executive Dashboard:** This sample shows a typical summary of project status for the Rational brand.

Shown is the CC/CQ/RTC/BF project schedules:

The idea here is the quick visual assessment. Ie. Red = Risk
Agenda

- Lean and Agile what’s behind the hype
- Lean & Agile Defined
- Before: Waterfall
- After: Agile
- Transformation
  - How we made it work
  - Lessons learned
- Tooling that made it possible
  - Rational Team Concert
Transformation
Agile and z/OS weren’t a perfect match

- Continuous integration
- A “shippable increment” every 2-4 weeks
- Small teams with interchangeable skills
- Deferred commitment and variable content
- Stakeholder feedback on a sprint basis
- Rapid reaction to changing requirements
- More frequent smaller releases
- User stories instead of line items
Cohesive team:
-- breaking down silos
-- dealing with disparate tools and processes
-- dealing with politics of different management chains

Getting Staffing in synch
-- How many testers per developer? What is the right sprint load for “done done done”

Multiple consumables
-- some things being done agile, some waterfall,
-- metrics and reporting requirements disparate

Sizings and pace:
-- what is the real pace/team velocity a team can sustain taking into account, meetings, etc that take place in a day? 6hrs?
-- how to correctly play planning poker by all team members, translation of points to measures of time
Prudently Agile – did what made sense

- Establish team view of agile
- Find ways to eliminate waste
- Identify different types of stakeholders
- Formalize the concept of “stretch” functions
- Improve estimation and planning
- Foster a “whole team” approach
- Continuously refine processes
A lean approach to unit test

A single set of testcases assists with two test tasks (FVT & UT)
Defect recording not required for UT bugs

Either Testers write all testcases, dev executes during UT or dev can help create testcases

FVT (Function Test) works in parallel with design and development
Variations/Test Definition included with design materials
Testcases developed in synch with code and executed as part of UT
Sprints are still defined at a team level usually revolving around a single development team and
Design: as per Lean/Agile tenants try to delay decision making: degree of design details and overviews
Whole Team: current definition of “whole team” function test and development and/or information
Definition of done will vary but must be defined by the whole team at Sprint planning meeting to
Discipline representatives: Development, Function Test, System Test, Level 2, & ID represented
Scrum team
  Scrum team members have tasks to do within the sprint
  Development & Function Test at a minimum
How to Create Product backlogs

Write product backlog items with different levels of detail:

- Fine grained for items about to be worked on
- Coarse grained for items further in the future
How to make a good story

Independent: dependencies lead to problems estimating and prioritizing.

Negotiable: stories are not contracts, leave or imply some flexibility

Valuable: to users or customers, not developers. Rewrite developer stories to reflect value to users or customers

Estimatable: because plans are based on user stories, need to be able to estimate them

Sized Appropriately: small enough to complete in one sprint

Testable: testable so that you have a easy what of knowing when finished. Done or not done
Factors Impacting Success

- Team size
- Team workload
- Strong management buy-in/leadership
- Team skill
- Tooling support
- Willingness for Process modifications
- Initial education on Agile and Tooling
- Determine new metrics
Lessons learned

- Different teams need different types of Leadership all need prioritization
- Change is hard
  - Need time, training to master new skills
  - Build your credibility
Agenda

- Lean and Agile what’s behind the hype
- Lean & Agile Defined
- Before: Waterfall
- After: Agile
- Transformation
  - How we made it work
  - Lessons learned
- Tooling that made it possible
  - Rational Team Concert
RTC built on the Jazz Platform is open and extensible
RTC provides integrated end to end support of any development process
RTC provides both planning and automated status to keep teams on track
RTC provides unique, in context, collaboration among software developers
You can adopt RTC in an incremental way using your existing artifacts
Needed a tool that had all the right elements

- Incremental Design
- Continuous Integration
- Test Driven Development
- Iteration/Sprints
- Backlog
- Learn and Adapt

- SCM
- Work Items
- Build
- Dashboard
- Process
## Rational Team Concert: A Closer Look

### Agile Planning
- Integrated release/iteration planning
- Effort estimation & progress tracking taskboards
- Out of the box agile process templates

### Project Transparency
- Customizable web based dashboards
- Real time metrics and reports
- Project milestone tracking and status

### SCM
- Integrated stream management
- Component level baselines
- Server-based sandboxes
- Identifies component in streams and available baselines
- ClearCase bridge, connector

### Work Items
- Defects, enhancements and conversations
- View and share query results
- Support for approvals and discussions
- Query editor interface
- ClearQuest bridge, connector

### Build
- Work item and change set traceability
- Build definitions for team and private builds
- Local or remote build servers
- Supports Ant and command line tools
- Integration with Build Forge

### Jazz Team Server
- Single structure for project related artifacts
- World-class team on-boarding / off-boarding including team membership, sub-teams and project inheritance
- Role-based operational control for flexible definition of process and capabilities
- Team advisor for defining / refining "rules" and enabling continuous improvement
- Process enactment and enforcement
- In-context collaboration enables team members to communicate in context of their work
Backlog Plan Mode

- Good for managing SCRUM backlog
- Support coarse & fine grained prioritization
- Ranking is reflected in all planning views
Iteration “Sprint” Planning

Understand how well you are progressing against your targets in real-time.

Plan and execute on iterations while managing load.

Drag-and-drop work items to change owners/create child parent relationships.
### Taskboards

#### Key Features:
- **Show stories linked to a set of associated tasks and their status.**
- **Drag and drop work items to change their state.**
- **See the work in progress or completed.**

#### Example Taskboard:
- **Title:** Improve Documentation for 4.4
- **Status:** In Progress
- **Details:**
  - Based on the symmetrical syntax, we should provide an altoview and the option to use it.
  - Two tests are passing, and one test is failing.
  - Two stories are in progress, and one story is completed.

#### Task Details:
- **Task 1: Rebase tests for 4.4**
  - Description: Rebase tests for 4.4
  - Status: In Progress
  - Progress: 1/2 completed

- **Task 2: Update documentation**
  - Description: Update documentation for 4.4
  - Status: In Progress
  - Progress: 1/2 completed
In context collaboration

Team Awareness
Shows team members and their online status
Shows what they are working on
SCM is stream and component based

- Work in parallel using streams to control sharing
- Easily add or remove change sets (tasks) from a stream
- Easily suspend and resume work
Work items capture traceability & effort

Predefined, custom and personal queries

Subscribe to work items you’re interested in

SCRUM built in artifact types

Understands and persists work items’ relationship to SCM and build artifacts

Integrated discussion threads & chat sessions

Query results
Builds - Extensible Continuous Integration

- Run personal builds to check your changes before sharing them with the team.
- Create build servers.
- Identify work items and change sets that went into the build.
- Historical view of the build queue with status.
- Even reconstruct a workspace from a failed build!
Transparency across disciplines and process
IBM Rational Software Conference 2009
Customized Dashboards

- Trending by project or by individual team
- All stories in current sprint
- Burndown charts
Contact Info

Email: zagonzal@us.ibm.com
Useful Resources

- **Rational Team Concert** (downloads, demos, info)
  

- **Agile Development**
  

- **Agility @ Scale: Strategies**
  
There are risks and costs to a program of action. But they are far less than the long-term risks and costs of comfortable inaction.

- John F. Kennedy

As quoted by May, 2007.
Backup
Jazz Platform Overview
The Software Delivery Platform – requirements for success

- Learn from industry mistakes
  - Assume integration around a repository
  - Design a data model for software engineering for the repository
  - Provide some sort of framework for tools to integrate around repository

- Take advantage of the Internet
  - Amazingly scalable and extensible
  - Integrates information on a massive scale
  - Collaboration on unprecedented scale

- Make it open and extensible
  - Data specified independently of tools
  - Tools (multiple) access data through HTTP/APP
  - Search and query through “structured indexes”, independent

Jazz Goal:
Be for collaboration tools
what Visual Studio and Eclipse are for the desktop
Jazz Foundation Services ….. What is it

- Server technology and a set of toolkits to build Jazz products
- Jazz team server is result of this effort
- Common infrastructure services
- Enabling technology for Collaborate, Automate and Report
- Enabler for Collaborative Application Lifecycle Management
- Helps drive consistency across products
  - Integrations
  - Common UI
  - Administration
  - Operating Environments
  - Scalability
  - Security

- Consume the foundation and build solutions on top of it
- Based on the Jazz Integration Architecture
- Allows products to focus on disciplines
  - e.g. RQM delivering a Quality Management solution on Jazz
- Leverage the frameworks to help software teams to
  - Collaborate on projects
  - Automate predictable tasks and processes
  - Report on status of the project, resources
- Rational Team Concert, Rational Quality Manager, Rational Requirements Composer and Future Products

Products Built with Jazz

Jazz team server is result of this effort
First wave of products built on Jazz technology

**Rational Requirements Composer**
Business Expert Collaboration
- Elicit, capture, elaborate, discuss and review requirements

**Rational Insight**
Cross-project and team reporting
- Performance management and measurement for integrated lifecycle intelligence

**Rational Team Concert**
Collaborative software delivery
- Collaborative SCM, work item, build automation & iteration planning

**Quality Manager and Test Lab Manager**
- Coordinate quality assurance plans, processes and resources

**Future IBM Capabilities**
- 3rd Party Jazz Capabilities
- Compliance & Security
- Performance management and measurement for integrated lifecycle intelligence

**OPEN SERVICES**
- Collaboration
- Presentation: Mashups
- Discovery
- Query
- Storage
- Administration: Users, projects, processes
Rational partner solutions extend the value of Jazz

- Validate business requirements visually for desired outcome
- Enable automated governance to insure compliance and minimize business risk around highly flexible and distributed development environments
- Estimates cost, effort, duration of projects and determines probability and inhibitors of success
- Rational partner solutions extend the value of Jazz
- Analyze project risk, technical debt and automatically set task level work breakdown structures
- Systems engineering and ERM goals with TOGAF process library
- Support heterogeneous environments with bidirectional integration with the JIRA change management system
- Allow bi-directional synchronization of defects and workflows
- Enables breakthrough development economics while minimizing the associated risks and challenges
- Enhance team collaboration with direct, per-project document linkages to Microsoft SharePoint and IBM Lotus Quickr libraries
- Eliminate risk of project failure with visual requirements definition through simulation
- Validate business requirements visually for desired outcomes
- Automated assessment and metrics for technical quality and team performance
Jazz Based Product Suite

**Rational Team Concert**
IBM Rational Team Concert is a team-aware software development platform that integrates work item tracking, builds, source control, and agile planning. Rational Team Concert interoperates with other products by providing Visual Studio integration and connectors for ClearCase and ClearQuest.

**Rational Team Concert for System z**
Rational Team Concert for System z provides distributed users with all of the capabilities of Rational Team Concert hosted on the robust System z platform.

**Rational Change Management Express**
IBM Rational Change Management Express exposes a subset of IBM Rational Team Concert to provide a robust collection of change management features, including work item tracking, process awareness and customization, team awareness, and project health viewing through team reports and Web dashboards.

**Rational Quality Manager and Rational Test Lab Manager**
Rational Quality Manager is a centralized test management environment that helps increase the efficiency and quality of software delivery through test planning, workflow control, testing and traceability, and metrics reporting. Rational Test Lab Manager, an extended component of Rational Quality Manager, helps to improve the efficiency of the test lab environment and optimize its utilization, cutting workload and saving on test infrastructure.
Jazz Based Product Suite

Rational Requirements Composer
IBM Rational Requirements Composer provides a platform for collaborative requirements definition that enables business analysts, client stakeholders and software development teams to elicit, capture, elaborate, discuss, review, and validate requirements using a variety of requirements definition techniques and collaboration capabilities.

Rational Project Conductor
IBM Rational Project Conductor is a project and resource management system optimized for software and systems delivery. It enables project and program managers to plan, schedule, and staff projects, with the right resources working on the right tasks. It provides management with control and visibility over project status and progress, and serves as the central repository for project and program data.

Rational Build Forge
IBM Rational Build Forge is a process execution framework that automates, orchestrates, manages, and tracks all the processes through each handoff within the software development lifecycle to create an automated software factory. Rational Build Forge integrates into your current environment and supports major development languages, scripts, tools, and platforms, allowing you to preserve your existing investments while adding valuable automation, acceleration, notification, and scheduling capabilities.