Agile Architecture
Changing Application Server

Veljko Krunic
Introduction

• Rally’s product migration to JBoss
• Wasn’t possible as a single release
  – Big architectural dependencies on “old” app server and portal
  – We wanted front end features that were not easy to do in portal
  – So we wrote a lot of custom code to support it
• “New” architecture on JBoss
• How to run a marathon and stay Agile
How to Run A Marathon

- Run slow in the beginning!
  - One person from the team

- Think!
  - “Just in time” architectural decisions

- But run!
  - Have a demo as early as possible

- At sustainable pace!
  - Biggest risk first
  - Ship new parts as soon as finished!

- Final sprint!
  - Whole team on JBoss!
Timeline

Legend
- Internal Release
- Market Release
Architectural Decisions

- Architectural decisions are the ones that would kill a project if done incorrectly
- Anticipation is better than reaction
  - Informal sessions with whiteboard
  - Risk focus
  - Find and prioritize technical risks
  - Quick prototyping to test architecture
- Postponed for the “last responsible moment”
  - Just in time
How Many Code Bases Do You See?

- Had to support JBoss and “old” app server
- Support from single code base
  - What if we ship “JBoss induced” bugs?
- Support from separate code bases
  - Difficult to synchronize
- Decision - single code base
  - Quality is the result of good processes and people
  - Continuous build system for both app servers
  - Osmotic communication through the code
- Allowed shipping new GUI features to market earlier
  - Incremental shipment of the new architecture
  - Where not dependent on JBoss app server
Initial Demo

- Only software that can be demoed is relevant
- Have a demonstration as soon as possible
- Demonstrate minimal features
- And demo again when there is new functionality
Sacrifice One

- Problem
  - Something is a low priority for the organization
  - But needs constant attention

- Solution
  - Initially, there is only one person on JBoss full time
  - The rest of the team is on the “old” appserver

- Consequences
  - Limits bandwidth spent on low priority tasks, yet the work is focused
  - Single code base = people change “old” app server all the time
  - “One” counterbalances them
  - The risk is that the knowledge is concentrated in a single team member
Technical Risks

- This was mainly a technically driven project
  - “Give me the same features on different technology”
  - With few new features
- We used risk driven scheduling
  - Backlog should include risks
  - Prioritize risks
- Identification of risks
  - Brainstorming
  - Early prototyping of complex issues
  - Test driven architecture
- Ranking
  - Include multiple people
Estimation

- This was a porting project
  - Port all we have to JBoss
  - What is “all”? 
- The problem is not the incorrect estimation of known things
  - The problem is finding scope
  - There was no automated way to discover issues
- Had to depend on people’s memory
  - Ask people
  - That are currently working on the “old” app server
  - And have to do a “context switch”
Ah… Estimation

- It is difficult to remember when not focused!
  - You are not working on it and have casual preparation at best
  - Even outside of meetings, focus is not sharp if you are working on something else

- Better estimation techniques didn’t really help
  - Your estimation is always bad if you forget the scope!

- This has some relation to problems in waterfall
  - If you do “architecture phase”, you are not intimately familiar with the problems in future

- Possible solution?
  - Put everyone on JBoss for a week, then estimate
  - People have more context
  - This time is a small portion of the total work in a large project
You are close to a release on “new architecture”

Should you put the whole team on it?

Options

- Develop new features on both app servers
- “Freeze” the “old” and work only on JBoss

Consideration

- Set up your plan to be cheap if you succeed!
- Even if you succeed, developing in parallel costs you money!
- So don’t do it!

What we did

- We implemented a few new features and “froze” that code
- Those features were enough for a legitimate release
- If we failed on JBoss, we would ship a “small release”
What Went Well

- Risk based scheduling
  - We didn’t have big technical issues in the previous two months
- Just in time architecture
  - We didn’t have to backtrack
- In container tests
  - A lot of bugs in porting projects are within the integration
- Single code base for “old” and “new” code
  - Didn’t induce bugs to shipped “old” releases
  - Saved work
  - Increased communication
- Nothing speaks like success
  - JBoss release was a success
  - Good quality and performance
  - New features
What We Would Do Better Next Time

· Whole team on JBoss for a week
  - Estimation would be better
  - People would know details of “new architecture” better
  - Presentations are not a substitute for practical experience

· “Sacrifice One” vs “Team per Task”

· “Sacrifice One” commitment
  - We had “next release” pressure
  - So we stopped working on JBoss for a few weeks
  - Single code line for both “old” and JBoss!
  - So we broke JBoss on few places
Next Release And Agile

· Next release pressure
  - Every organization is focused on the “next release”
  - Agile organizations release all the time!
  - Waterfall once in a while!
  - So agile organizations feel this pain more often

· Agile organizations need to be better in controlling “next release” overfocus
  - Think time, not releases
  - What is the impact of this decision in next six months?
Questions?