Practical Iterative Development

Kostas Flokos
Technical Manager
kfl@upcom.be

Presentation Summary

- Why a methodology?
- Explain concepts like “iteration”, “phase”
- Demonstrate importance of “iteration”
- Provide practical answers to typical questions
- Relate to java development

Why a methodology?

- Answer : in order to achieve QUALITY
- Information Technology projects have serious Quality issues
  - Almost 50% never make it to the end-user
  - Of them, more than 50% are over budget
  - etc.

Are methodologies enough?

Standards believe so

- ISO 9000 series
- CMM
- Etc

Reality is that

- Procedures without knowledgeable professionals do not reach the goal
- Knowledgeable professionals without procedures is frequently worse
- Lots of procedures slow down the performance and decrease the commitment of the individuals
- A good combination of the two is the solution

Methodology advantages

- Control complexity
  - all methodologies suggest splitting the work in easier to manage blocks
- Allow planning
  - putting the blocks together with their duration gives us an estimate for the delivery
- Force steps
  - written requirements are frequently forgotten in ad-hoc development
- Permit continuation
  - documentation and other internal deliverables help the next team continue the work of the previous

Main concepts

- Phase
  - Part of the project during which a major risk of the project is resolved
    - In the Rational Unified Process it is related to the common risks identified
      - Business Understanding – Inception
      - Technical Understanding – Elaboration
      - Quality Development – Construction
      - Error-free deployment - Transition
- Iteration
  - Part of the identified functionality implemented as a small project

Why iterations?

Purpose:

- Reduce project risk
- Late risks identified early
- Understand required quality early
- Have something that runs very early in the project

What:

- Frequent incremental deliveries
- Early client's feedback
- Continuous interaction with all project participants

Iterations – How

- Treat an iteration as a project
  - Delivery may go to Production
  - Full development cycle completed
    - Analysis, Design, Coding, Testing, Deployment
- Iterate regularly
  - At most every 8 weeks
  - Iterate as soon as sufficient functionality may be delivered
- Review and re-plan at the end of every iteration

Planning guidelines – 1

- Break the system in functional blocks (Use Cases)
  - A Use Case has a tangible result for the end-user!
- Always plan based on Use Cases
  - Even if the project is stopped in the middle, parts of functionality might be made available to the end-users
- Attack risky Use Cases first
  - If the risk manifests itself it will be easier to control early in the project

Planning guidelines – 2

- Pay attention to dependencies between Use Cases
  - Even if the “update” is riskier in a project than the “create”, you will not be able to test the “update” without the “create”!
- Do NOT plan based on technical layering
  - Functionality may not be clear or complete when lower layers are developed
  - If the project is stopped in the middle, nothing may be made available
- Add contingency in form of iterations during the Transition
  - Absorb during those iterations either small change requests or false initial estimates

Planning guidelines – 3

- During early phases (Inception & Elaboration)
  - Base the planning on the Vision or other High Level functional document
  - Create the list of Use Cases to be detailed
  - Make a plan on the UCs detailed per iteration
- During later phases (Construction & Transition)
  - Do not start them if high level functionality is not known
  - Dependencies must be clearly understood
  - Overall architecture agreed
  - Plan based on complete list of UCs, grouped per iteration

Follow up guidelines – 1

- Keep a fixed duration per iteration
  - Based on metrics from earlier iterations, it is possible then to correctly estimate promised functionality
- Do not accept changes during an iteration
  - If changes are accepted, team is disorganised and PM can no longer understand diversions from estimates
- Closely follow team work
  - Daily or bi-weekly status meetings are necessary to assess the iteration status

Follow up guidelines – 2

- Make sure iteration is complete
  - Complete is not 99%; it is 100%
  - All deliverables should be completed
    - Use Cases drafted and agreed with business users
    - Code completed and unit tests cover it and succeed
    - Test Cases cover all Use Cases
    - Test Scenarios correspond to all paths in UCs and execute successfully
    - End-users may use the partial application without problems

Best practices – 1

- Automated and unit testing ARE necessary
  - They are the only way to make sure an iteration is successful
  - The same tests must be run again and again; if they are not automated, the overhead is enormous
- Manage changes : code and functionality
  - If during development, important functional change is discovered, plan it for another iteration
  - During acceptance, you might have to return to previous iterations
  - Even documents should be version controlled; an iteration may implement an early version of a UC and another iteration the final version of the UC

Best practices – 2

- You MUST finish at the end of an iteration deadline
  - Time should not slip, so that you have an indication of time required to complete tasks
- Re-plan continuously if required
  - If part of functionality does not qualify for Production, re-plan it for a future iteration
- Cross-check deliverables
  - All deliverables should have a next step to control them, like Functional Specifications by development and testing

What if? #1

- Original estimates are proven wrong!
  - Iteration estimates wrong
    - Remove functionality from scope
    - Deliver completely what is possible
    - Plan the left over for later iterations
  - Project estimates wrong
    - It should be reflected to individual iterations
    - Early on an indication of delay should be presented
    - Continue completing iterations
    - Do NOT deliver incomplete functionality (without testing for example)
    - Negotiate for less functionality, more people or both!

What if? #2

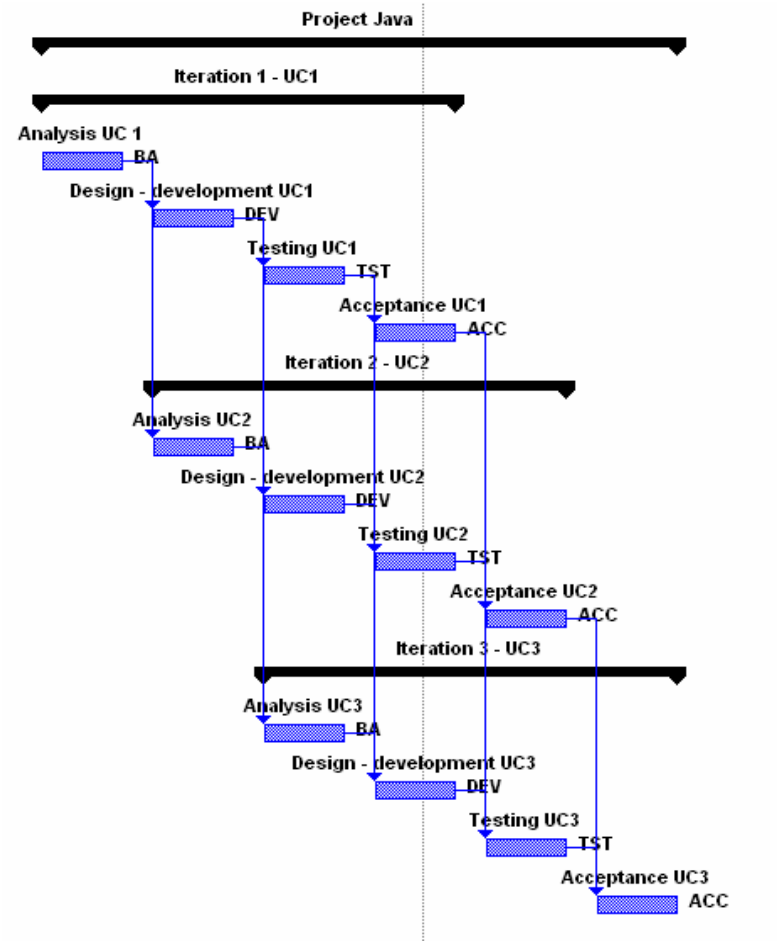
- “We are that close to the end, but haven’t finished yet”
  - The 99% syndrome
  - Close the iteration when it was agreed and consider in its scope only the completed (100%) functionality
  - Plan the 1% incomplete for subsequent iterations
  - Do NOT postpone an iteration end; it might never finish

What if? #3

- “Use Case x is not complete yet, but we all know what is required and the client agrees with it; should we start development?”
  - Be VERY sceptical; written documentation may not be contested, but ...
  - Users may change their mind or pretend they did not understand the text
  - Start the development only if you are certain all agree to work on incomplete specifications

What if? #4

- "I have 3 sub-teams in my project (analysis, development, testing); in an iteration each sub-team is idle while the others are working!"
  - Perform the analysis work in a previous iteration than the development and testing
  - Allow one iteration for acceptance testing, while the rest of the team continues with the next functionality



What if? #5

- "My project is part of a bigger project with different deliverables that must be integrated?"
  - Welcome to the real world!
  - Do not leave integration effort for the end
  - Integrate regularly, even with dummy versions!
  - Insist on adhering to the principle of complete deliveries and demand other components to offer the final functionality you depend on
  - Plan your iterations based on the final functionality offered by the others

What if? #6

- “Our project goes quite well, but we will not make it. The delivery date is fixed and we can’t compress the tasks anymore.”
  - Talk to client and let him decide
  - Offer him options to choose from; not just “we’ll be late, sorry”
  - Frequently dates are not written on stone
  - If forced to deliver, continue working after the delivery till project is complete; the client will always give you a second chance – even with penalties

What if? #7

- One of your core team members stays sick for a long period of time
  - Typical risk management techniques
  - Exploit the contingency iterations to postpone some lower level functionality

Typical mistakes! #1

- Users do not participate in the project
  - Development based on Use Cases agreed
  - Users do not understand Use Cases or other Functional Specification documents!
  - Only by seeing the final product, users may make final comments
- Overall quality is not similar to final delivery!
  - Deployment scripts, installation and user guides are frequently missing
  - If system has to go in Production, team responsible for it will complain
  - Frequent deliveries require not only installation, but upgrade instructions as well
  - System is not ready for later releases

Typical mistakes! #2

- Draft versions are considered as early iterations!
  - Delivery can NOT go Production
  - No indication of the real work required to complete the task
- Acceptance team does not control deliveries
  - Never ready to put something in Production
  - Final acceptance takes too long
- Change requests are accepted during the iteration
  - Impossible to assess team performance
- Developers tend to wait till final agreed specifications exist
  - Change is not part of their culture! They will have a problem with acceptable changes in the project

Iterations in Java

- Implications in Java development
  - Unit test extensively
    - jUnit or similar
  - Design for unit tests
    - Spring or similar
  - Automate functional testing
    - HttpUnit or similar
  - Control change
    - CVS or similar
    - Bugzilla or similar
  - Automate builds
    - Ant, Maven or similar
    - Cruise Control

Spring

- Started as a micro-container
  - To support the Inversion of Control pattern
  - Evolved into a powerful framework
- Concepts in Spring are very important for Iterations
  - Increase decoupling between classes
  - Helps unit testing
  - Improves quality

Spring example

Typical Inversion of Control example

- Before

```
public operation (params){
    Connection conn=getDatabaseConnection(hostname, ...);
    ...
}
```

- After

```
public void setConnection (Connection conn){
    instanceConnection = conn;
}
public operation (params){
    Connection conn = instanceConnection;
    ...
}
```

Spring example - Simplified unit testing for DAOs

- Instead of
  - Trying to find a way to get this precious database connection
  - For every operation
    - Start a transaction
    - Invoke the operation
    - Rollback the transaction
  - Try to understand whether the test succeeded or failed
- Do
  - Extend from `AbstractTransactionalDataSourceSpringContextTests`
  - Get the instance to the requested object
  - Invoke the operation
  - Check the results

Conclusions

- Iterative development improves rate of success in IT projects
- Developers and end-users are partners in that effort
- Java development initiatives assume or support iterative development
- Use of sound design patterns greatly helps iteration-based projects

More information

- Agile software development
  - http://en.wikipedia.org/wiki/Agile_software_development
- Scrum
  - http://www.controlchaos.com/
- Rational Unified Process – Method Composer
  - http://www-306.ibm.com/software/awdtools/rup/index.html

Questions?