Software Engineering
G22.2440-001

Session 12 – Sub-Topic 1
Risk Management in Adaptive Software Engineering

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

• Developing Risk Response Strategies
• Risk Management in Agile Processes
• Agile Project Planning
Part I

*Developing Risk Response Strategies*

See: [http://alistair.cockburn.us/crystal/talks/apms1/advancedpmstrategies1-180.ppt](http://alistair.cockburn.us/crystal/talks/apms1/advancedpmstrategies1-180.ppt)

---

Part II

*Risk Management in Agile Processes*

See: [http://alistair.cockburn.us/topicalindex.htm](http://alistair.cockburn.us/topicalindex.htm) (management topics)

*The Role of the Manager in Modern Agile Projects*
PM Skills Needed

• Traditional engineering skills:
  – plan, analyse, design, build, test, install etc.
• Understand the importance of documenting and signing off the analysis and design before building/coding.
• Prior Experience:
  – Does this approach match the real world?
  – Is this idea universally applicable?
  – Are there valid alternatives?

Extreme Programming (XP)

• One of a group of “agile methods”.
• Created to handle problem domains whose requirements change:
  – Customers may not have a firm idea of what the required system should do. Indeed the system's functionality may be expected to change every few months.
  – In many software environments, dynamically changing requirements is the only constant.
Extreme Programming (XP)  
(continued)

• XP emphasises team work.
• The method encourages four values:  
  – communication;
  – simplicity;
  – feedback; and
  – courage.
• Note: XP is as an example of agile methods. Some of XP’s effects on project management are shared by DSDM, SCRUM etc.

XP Challenges Assumptions

• XP says that analogies between software engineering and other engineering are false:
  – Software customers’ requirements change more frequently;
  – Products can be changed more easily;
  – The ratio of design cost/build cost is much higher if we consider coding as “design” and compile-link as “build”:
    • the “build” task is so quick and cheap it should be considered instant and free,
    • almost all software development is “design”.
XP Challenges Assumptions
(continued)

• The design meets known existing requirements, not all possible future functionality.
• Beck (1999): “If you believe that the future is uncertain, and you believe that you can cheaply change your mind, then putting in functionality on speculation is crazy. Put in what you need when you need it.”

How XP works

• As with RAD and DSDM etc. programmers meet and communicate with customers regularly, and the software gets released incrementally.
• Programmers always work in pairs (considered more productive).
How XP works
(continued)

- Testing is the start point, not the end:
  - For each user story, the customer first writes an
    acceptance test.
  - For each unit the programmer writes a set of unit tests.
  - Then each unit in a story is coded.
  - When a unit is ready, its tests are run automatically.
- Customers are allowed to suggest improvements.
- Redesigns are common – they are part of refactoring - and handled easily.

What Effect Does XP Have On Project Management?

- Project scope
  - XP embraces changes in requirements. There is little value in the traditional project scope that forms a contract for what functionality will be provided for a fixed cost/effort.
  - Instead, the customer writes on cards a set of user stories that describe what they want the new application to do for them.
What Effect Does XP Have On Project Management?
(continued)

• Task and project estimating
  – The time (duration) to complete each user story is written on the relevant card. If the story will take more than 3 weeks, it must be split into two or more smaller stories.
  – These stories are prioritized in a release plan; the plan sets out which stories that will be produced in each iteration. XP projects may have fixed time (scope varies) or fixed scope (time varies), but not both.

What Effect Does XP Have On Project Management?
(continued)

• Quality management
  – For each user story the customer must write an acceptance test. These are produced before any software is written to satisfy the story. Acceptance tests are normally automated so they can be used as regression tests each time the software is changed.
  – Programmers always work in pairs to ensure learning, adherence to standards, and avoidance of errors.
What Effect Does XP Have On Project Management?
(continued)

• Progress monitoring
  – Progress monitoring can be as simple as keeping a track of which story cards have been marked as completed (passed its acceptance test).
  – The XP concept “project velocity” (the speed at which the developers are producing work) prevents unwarranted optimism!
    * E.g., “The first task took too long, so I expect the other tasks will be done more quickly than expected.”

What Effect Does XP Have On Project Management?
(continued)

• Change control
  • XP is intended for environments where requirements are expected to change several times before the project is over.
  • As new user stories are programmed, changes to existing work may be needed.
    – XP developers put a lot of effort into improving code and keeping it as simple as possible (refactoring).
    – Constant refactoring necessitates regression testing of all affected modules. This reinforces the need for automated testing (see Quality Management above).
What effect does XP have on project management?
(continued)

• Risk management
  – Traditional software development focuses on reducing risk from software errors (verification) by having copious plans and procedures.
  – XP focuses on reducing risk by ensuring customers' requirements are met (validation).
  – Barry Boehm (2002) suggests that the best way to decide whether XP is suitable for a given project is to assess risk exposure.

Bibliography

Agile Manifesto (2001)
  http://www.agilemanifesto.org/
  http://www.extremeprogramming.org/
Part III

Agile Project Planning

Requirements Specification

• Usage Scenarios
• Use Cases
• User Stories
• Others…
Usage Scenarios

• Sequence of steps describing an interaction between user and system
• Typically focus on user interface
• Included in use cases, possibly appended to user stories

Use Case

• ID: short name
• Actor
• Precondition, trigger
• Success and failure end conditions
• Main scenario
• Extensions, variations
• Includes and included-by
How to Develop a Use Case Model
(review)

1. Identify who is going to be using the system directly - e.g., hitting keys on the keyboard. These are the Actors.
2. Pick one of those Actors.
   • Example: Sales Order Clerk

3. Define what that Actor wants to do with the system. Each of these things that the actor wants to do with the system become a Use Case.
4. For each of those Use Cases decide on the most usual sequence of steps when that Actor is using the system. What normally happens. This is the main scenario.
How to Develop a Use Case Model
(review)

5. Describe that main scenario for the use case.
   – Describe it as "Actor does something, system does something. Actor does something, system does something."
   – Only describe things that the system does that the actor would be aware of and conversely you only describe what the actor does that the system would be aware of.

Main Scenario Example

Enter a Sales Order
1. Sales Order Clerk enters the customer’s last name.
2. System displays all matching customers.
3. Sales Order Clerk selects one of those customers.
4. System displays that customer’s details.
5. For each item that the customer wishes to order, Sales Clerk enters the line details.
6. When all line items have been entered, System confirms the order.
How to Develop a Use Case Model
(review)

6. Now consider the alternates and add those as extending use cases.
   • Example: In use case Enter a Sales Order,
     1. If System finds no customers matching the last name, System displays an error message.
     2. System allows Sales Order Clerk to enter a different last name to search against.
   • New Customer Not Found use case extends Enter a Sales Order use case

How to Develop a Use Case Model
(review)

7. Review each Use Case description against the descriptions of the other Use Cases.
   – Notice any glaring commonality?
   – Extract those out as your common (included) Use Cases.
Another Main Scenario Example

Credit Check a Customer
1. Sales Order Clerk enters the customer’s last name.
2. System displays all matching customers.
3. Sales Order Clerk selects one of those customers.
4. System displays that customer’s details.
5. System also displays customer’s payment history for past six months.

Example Includes

- New Display Customer Details use case
  - Enter a Sales Order use case includes Display Customer Details use case to lookup and display a customer’s details.
  - Credit Check a Customer use case includes Display Customer Details use case to lookup and display a customer’s details.
- Note: Customer Not Found use case extends Display Customer Details use case
How to Develop a Use Case Model
(review)

8. Repeat steps 2 - 7 for each Actor.
   • Tips:
     – Don’t make use cases too small, limit decomposition into extending and included use cases
     – Don’t need to include all details, capture only functional requirements

User Stories

• Alistair Cockburn describes as a Use Case at 2 bits precision (goal, main scenario)
• Says Use Case typically 4 bits precision (failure conditions – extends, modularity - includes)
Planning with Use Cases or User Stories

- Drive creation of acceptance tests
- Construction of engineering tasks
- Derivation of time estimates for release plan

Time Estimation

- Based on experience
  - First iteration is guestimate from previous projects
  - Second and later iterations consider velocity within this project
- Task breakdown
  - Use cases or user stories may require development of underlying infrastructure, e.g., database
- Consider differences in task difficulty
XP Time Estimation

• XPers rate user stories as 1-3 “Perfect Engineering Weeks”
  – If estimate is longer than 3 weeks for one story, break the story down into multiple stories
  – If estimate is less than one week, combine stories
  – Estimate only in weeks, not months

Ideal Development Time

• Usually w.r.t. one programmer, or one pair
• Time it would take to implement if there were no distractions, no other assignments, and you knew exactly what to do
  – No phone calls to take, no meetings to attend, no questions to answer or get answered, no web surfing, …
Time Estimation Problem

- Excellent accuracy estimating relative difficulty
  - More experience -> more accuracy
  - Compare to “similar” stories (or use cases) for existing code
- Very poor accuracy estimating elapsed time
  - Interruptions, meetings, etc.
  - Individual differences
  - ???

Time Estimation Solution

- Time estimation is scary and difficult
- Therefore move as quickly as possible to estimating by comparison
  - Story Estimation Units
    - $s$
    - Points
- Estimate resources, estimate velocity
- Only then convert to time duration
Release Plan Meeting

- Managers, developers, and customers must be present
- Estimate how long it would take to complete each user story (or use case)
- Prioritize the stories (or use cases) in terms of importance
- Pick set of stories (or use cases) to be implemented for next release
- Developer team arranges into iterations
XPers Have Frequent Releases

• Frequent, small releases - each one with more and more functionality
• Critical in getting early feedback from customer
• Helps track total amount of work done during each release and the iterations leading up to that release
• Keeps customer happy
• Raises developer morale

XP Iterations

• Numerous iterations leading up to each release
• Each iteration is 1-3 weeks
• Things to be done in each iteration are planned out in detail just before that iteration
• User stories to be implemented in this iteration are called engineering tasks
• May include “failed” stories from previous iterations
• Developers sign up for the tasks
• Do not plan [further] ahead!
Iteration Planning

- About one-half day per iteration
- Select user stories (or use cases) to implement
- Developers
  - Ask questions to clarify understanding
  - Determine workflow and screenflow
  - Create Engineering Tasks
  - Estimate tasks (revise/refine user story or use case estimates)
  - Sign up for tasks (promotes ownership)
- Balance load, get to work

Iteration Plan with Cards
Iteration Plan with Cards
(after First Iteration)

101
H 1

167
H 2

231
H 1

151
H 2

134
M 1

165
M 1

234
H 1

132
H 1

173
M 2

45

Iteration Plan with Cards

101
H 1

167
H 2

231
H 1

151
H 2

134
M 1

165
M 1

234
H 1

132
H 1

173
M 2

46
Iteration Plan with Cards

Iteration Tracking

- XPers do approx. twice weekly
- Ask each developer, for each task:
  - How much work did you get in on it
  - How much work is there to go
- Note increasing estimates
  - Possibly too complex, break down
- Note inability to get much time in
  - Possibly too much overhead, reduce
- Feed forward into next iteration plan
Planning in XP Lifecycle

Relative Timing