



The Software Development Process

Tom Fulton

Mentor Communications and Training, Inc.



Outline

- The Rational Unified Process (RUP)
- Extreme Programming (XP)



Existing Processes

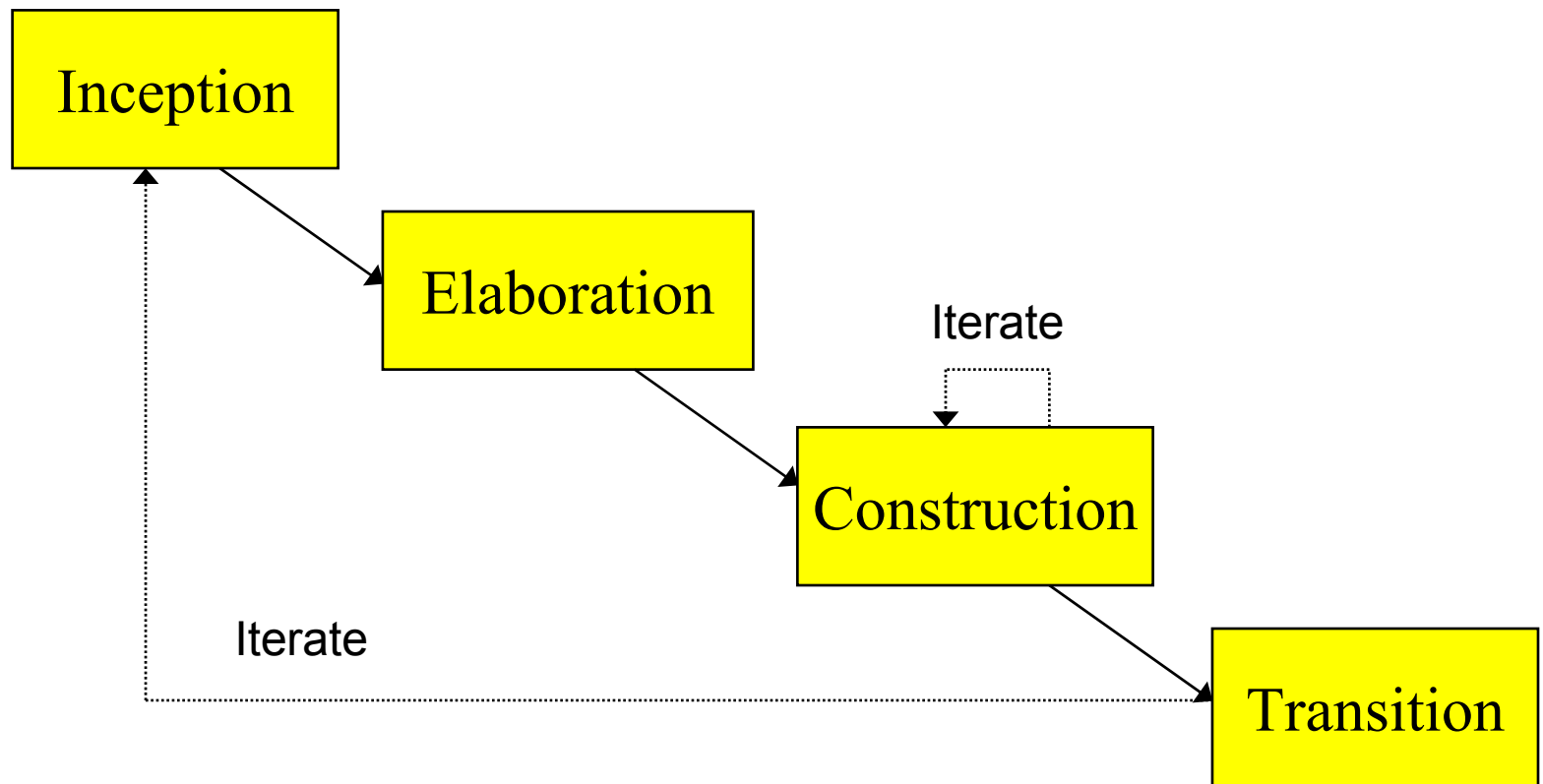
- Project management ideas in the software industry are represented by two contrasting schools of thought:
 - Structured processes like the Rational Unified Process (RUP or just UP)
 - Agile methodologies like Extreme Programming (XP)



The Rational Unified Process

- Created by “The Three Amigos”
 - Grady Booch
 - James Rumbaugh
 - Ivar Jacobson
- Their book is *The Unified Software Development Process*, Addison-Wesley, 1999.
- Our take is based on Fowler’s *UML Distilled* book, 2nd edition
- RUP is an extensive and detailed process
- Note that the UML is independent of process

Process Overview





Process Overview

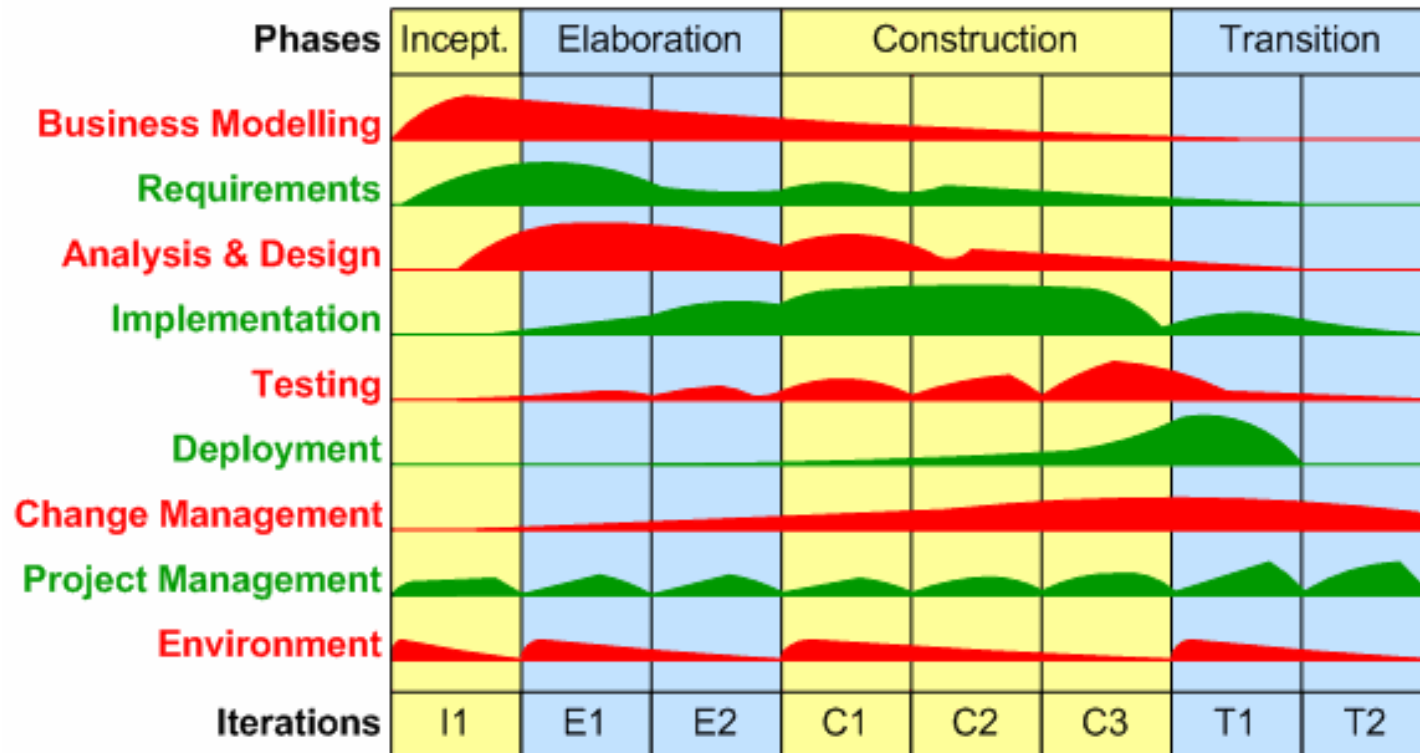
- Four stages:
 - Inception
 - Establish business case
 - Decide on scope of project
 - Elaboration
 - Collect detailed requirements
 - High level analysis
 - Design baseline architecture
 - Create construction plan



Process Overview

- Construction
 - Build the software
- Transition
 - Beta testing
 - Performance tuning
 - User training

RUP Processes





Inception Phase

- Work out business case for project
 - Rough cost estimate
 - Rough profitability estimate
 - Some sense of project scope, which might involve some initial analysis
 - Very brief phase; just enough to get sponsor's willingness to do a deeper look at the project



Elaboration Phase

- Driven by *risks assessment*
- There are four categories of risks
 - Requirements risks
 - What are the requirements for the system?
 - Is it the right system for the job?
 - Does it do what the customer needs?
 - Technological risks
 - Can you select the technology?
 - Will the pieces fit together?



Elaboration Phase

- Skills risks
 - Can you get the staff and expertise you need?
- Political risks
 - What political forces will seriously affect the project?
- Additional risks may be exist, but these categories almost always are present



Requirements Risks

- Determining the project requirements
- UML **use cases** are ideal here
 - Help evaluate the domain model
- This stage is similar to the requirements capture phase in PMI
- Formulate the **domain model**
- Modeling techniques
- Everything should be very high level
- Developing prototypes



Technological Risks

- Logical place for multiple prototypes
- Some of the biggest risks are where different components fit together
 - Object / database interface very common
- Architectural design decisions
 - Identify major components
 - How they will be built
 - Focus on areas difficult to change later



Skills Risks

- Get training where needed
 - Helps avoid mistakes
 - Way to acquire specific knowledge
- Mentoring
 - Can significantly shorten the learning curve
 - Builds your own skills while producing
- Monitor the literature, web sites, etc.
 - Design patterns
 - New language developments



Political Risks

- Fowler: “I can’t offer you any serious advice on this, because I’m not a skilled corporate politician. I strongly suggest that you find someone who is.” 😊
- Somewhat more practical advice:
 - Best practices
 - Honest estimates
 - Both budget and time – identify problems as early as possible
 - Iterative and incremental deliverables
 - Regular communication with customers



Elaboration Phase

- Planning construction
 - Categorize use cases by business value: high, medium, and low
 - (Fowler: "It's considered bad form to put everything in the 'high' pile.")
 - Categorize use cases by development risk: high, medium, and low
 - Identify difficult, big impact, and uncertain elements
 - Developers estimate time for each use case
 - Now you can make a plan, based on risks and priorities



Elaboration Phase

- Planning construction
 - Decide on iteration length
 - Fixed length helps you get into a rhythm
 - Consider how many developers you have, and how much of their time when you have them
 - Assign use cases to iterations
 - Do the highest risk items first!
 - This is both obvious and counter to human nature



Elaboration Phase

- Allocate time for tuning, documentation, and packaging for delivery (10 to 35% of the total)
- Add in a contingency factor
 - 10 to 20% of the construction time, depending on how risky things look
- Create a release plan
 - Shows which use cases done in each iteration
 - Not cast in stone, but does represent a commitment between developers and users



Finishing Elaboration

- When is elaboration finished?
 - Developers feel comfortable providing estimates for each use case
 - All significant risks have been identified, with plans on how to deal with them
 - A construction plan has been formulated



Construction Phase

- Build the system in a series of iterations
 - Each iteration is a mini-project, with analysis, design, coding, testing, and integration
 - Each iteration finishes with a user demo and system tests to confirm correctness
- Each iteration builds on use cases from the previous iteration
- Each iteration revises some existing code



Refactoring

- Programs grow as new, not necessarily anticipated functionality is added
- Can eventually lose coherence
- Solution: redesign the existing program to better support changes → *Refactoring*
 - Much reluctance to do this (“if it ain’t broke, don’t fix it!”)
 - Avoiding results in increased complexity, which will eventually crash



Refactoring

- Techniques to reduce the pain of redesign
 - Tiny steps
 - Rename a method
 - Move fields from one class to another
 - Consolidate two similar methods into a superclass
 - Each step is small, but a couple hours is worth a lot to a program



Refactoring

- Principles:

- Don't refactor and add functionality at the same time
 - Impose a clear separation – even on an hourly basis
- Make sure you have good tests in place before you begin refactoring
- Take short, deliberate steps
- Do a little every day



Integration

- Integration is the process of incorporating changes into existing code
- Should be done very frequently, with a complete test place in place
- Best practice:
 - The Daily Build – rebuild the entire code every night and run all the tests



Transition Phase

- Bringing project from development to completion in a given iteration
- This is when you do optimization
- No new development to add functionality, unless small and absolutely essential
- Development allowed to fix bugs
 - Time between beta and final release is an example of a transition phase



Extreme Programming



Extreme Programming

- Abbreviated XP
- Example of a “lightweight” methodology
 - Not as formal
 - Few rules and practices
 - Generally easier to follow
 - Much less emphasis on design and documentation
- Formulated by “The Three Extremos”
 - Kent Beck
 - Ward Cunningham
 - Ron Jeffries



Extreme Programming

- Collection of existing, simple practices
 - Major dedication to testing
 - Paired programming
 - Frequent refactoring
 - Simplest possible solutions
 - Extensive customer involvement
- Focuses on writing code
 - Kent Beck: “Documentation, design, formal review, separate QA; it’s all a waste of our time”



Testing

- Formulate tests first
 - Not allowed to write code until tests in place
 - Tests should be automated and, if possible, embedded directly into the software
- Two types:
 - Unit tests for each specific class
 - Functional tests to check entire system
 - Functional tests generally run every night
- All tests must be running at 100% before a developer releases their code



Testing

- Ironically, extensive tests give developers confidence to change things
 - If the tests work, the code is fine
 - Can make incremental changes and re-run
 - Not afraid to break existing functionality, because you can tell exactly what's wrong and where



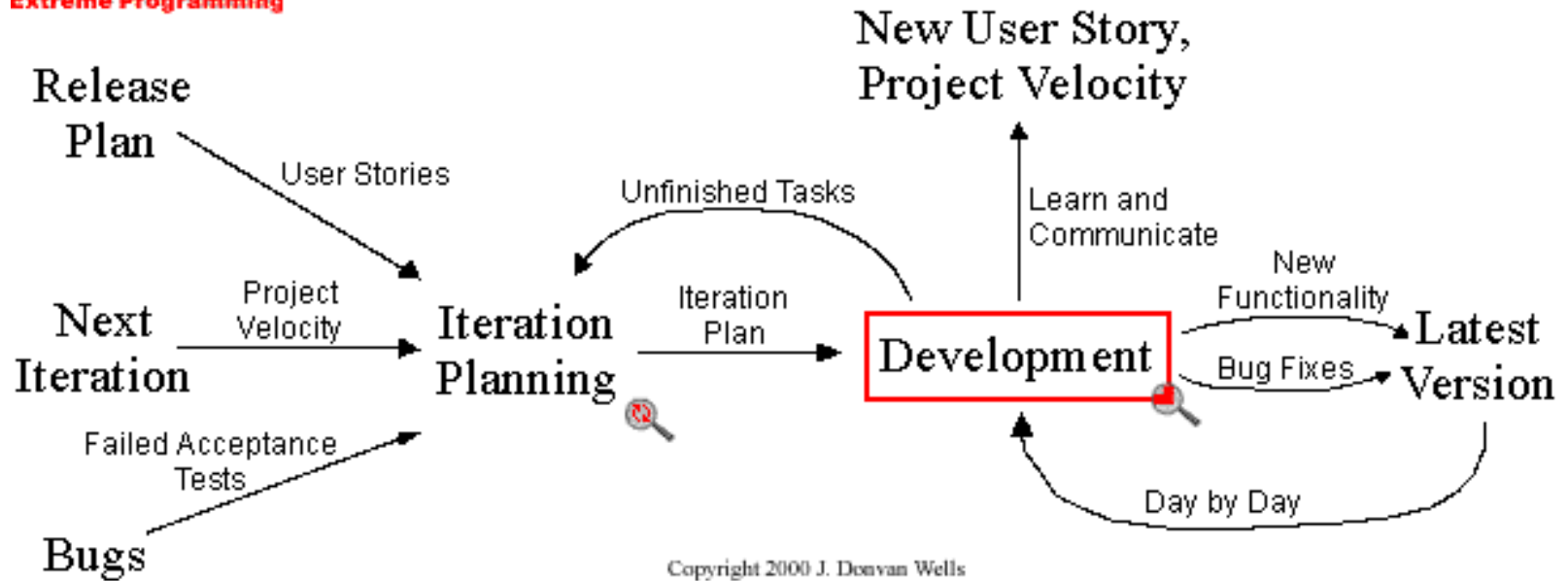
User Stories

- XPers call use cases “user stories”
 - Idea is to emphasize the customer’s view of the project
 - Short, couple of sentences description of what the customer wants the software to do
 - Take the place of large requirements documents in more formal methodologies
 - Each one should be about a week’s worth of effort
 - If not, break it into smaller user stories
 - Note similar recommendation in PMI

Extreme Programming



Iteration





Paired Programming

- Programmers must work in pairs
- XPers believe that two programmers working together are more than twice as productive as one
- One programmer generates code while the other thinks about the design and tests
- Then they switch



Paired Programming

- Code belongs to the entire group
 - Helps break proprietary fiefdoms
 - Programmers are allowed to change any code anywhere in the system, as long as the tests still run afterwards
 - Must be a strict adherence to common coding standards by all the developers in the project



The “Planning Game”

- Planning done as a game
- User stories are the pieces
- Developers and customers are the players
- Goal is a release plan stating which user stories will be implemented in each release and the dates for the releases



Other Principles

- Refactor regularly
- Always look for the simplest possible solution to any problem
 - Avoids building in functionality that “might be needed later”
 - Based on the idea that with a complete set of tests available, developers won’t be afraid to later make extensive modifications to the code



Other Principles

- Onsite customer
 - Customer available for formulating user stories
 - Answers questions later
 - Resolves disputes
 - Sets smaller priorities
 - Note: Sometimes it is difficult to get the customer to agree to this, but it's fundamental to the process



Summary

- The Rational Unified Process (RUP)
 - Inception
 - Formulates business case and scope
 - Elaboration
 - High level analysis
 - Risks assessment
 - Construction
 - Build the software
 - Transition
 - Debugging
 - Performance tuning



Summary, continued

- Extreme Programming (XP)
 - Strong focus on testing, both unit and functional
 - Paired programming
 - Customer part of the team
 - Team ownership of all code
 - Simplest possible solution is used
 - Regular refactoring



References

- Belanger, T.C., How To Plan Any Project, 2nd Edition, 1995.
- Fowler, M., UML Distilled, 3rd edition, Addison-Wesley, 2003.
- Kruchten, P., The Rational Unified Process: An Introduction, Addison-Wesley, March 2000.
- Jacobson, I., et al., The Unified Software Development Process, Addison-Wesley, February 1999.
- McConnell, S., Rapid Development, Microsoft Press, June 1996.
- Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, October 1999.
- Beck, K., and Fowler, M., Planning Extreme Programming, Addison-Wesley, October 2000.
- Jeffries, R., et al., Extreme Programming Installed, Addison-Wesley, October 2000.
- Fowler, M., Refactoring: Improving the Design of Existing Code, Addison-Wesley, June 1999.