Xtreme RUP

by

Net OBJECTIVES

Lightening Up the
Rational Unified Process

Agenda

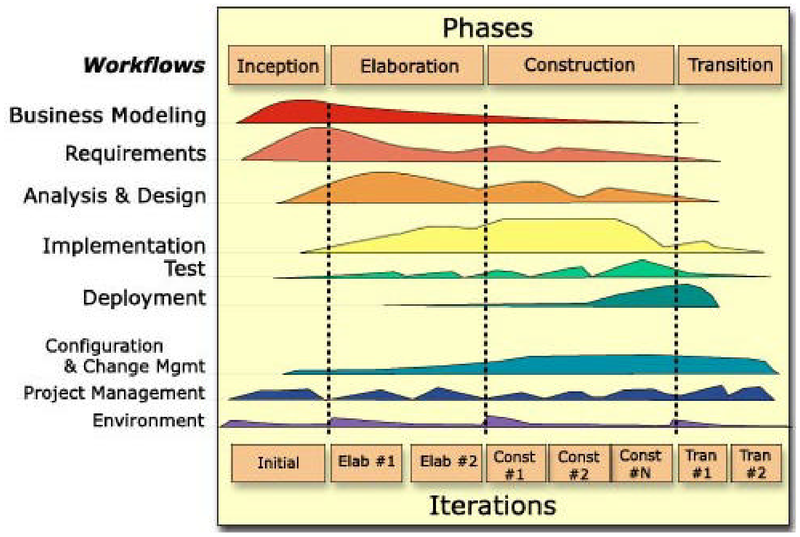
- RUP Overview
- Typical RUP Challenges
- Xtreme Programming Paradigm
- Document driven or results driven?
- Integrating RUP with XP to avoid "paralysis by analysis"

RUP Summary

- Purpose of RUP
  - A software engineering process
  - Create a repeatable methodology for achieving high quality results
- Design and delivery of RUP
  - Can adapt and tailor to fit your needs
  - Select artifacts desired
  - Iterate to level needed
  - Create a well-defined process for your organization
- A process product
  - Use it to develop a process of software development

2/9/2001 Copyright © 2001 Net Objectives 3

Two Dimensions of the RUP



2/9/2001 Copyright © 2001 Net Objectives 4

RUP Approach

- Iterative - allows for changing requirements
- Mitigate risks early
- Integration in stages
- Developers learn along the way
- Process can be improved along the way

RUP Steps

- Inception
  – determine scope of the project and create business case for it
- Elaboration
  – do requirements analysis
- Construction
  – build through iterations of analysis, design, implementation and testing
- Transition
  – beta testing, documentation, training

RUP Paradigm

- Stresses communication between teams
- Finding problems up front result in big savings
- Document (artifact) driven
- Use-case driven
- Role driven
- Component-based architecture

RUP's Focus

- Assumes you can get useful, detailed information in requirements
- Assumes costs rise dramatically the later changes are introduced into the system
- The focus is on getting as good information as possible as soon as possible

Challenges That Show up in the Field

- Paralysis by Analysis
  - getting more information than can be used
  - losing site of the big picture
  - gathering information that ends up not being used (e.g., details for discarded features)
  - no one's heard of the 4th little pig because the wolf ate him while he was planning his plan
- Detailed information gathering can delay date of first iteration
  - delays feedback
  - results in greater investment than needed

Do We Drive By Plan or By Feedback?

- Some balance between detailed planning and 'driving by feedback' must be achieved
  - too much planning is ineffective - you never start
  - not enough planning is ineffective - you never finish (you're always changing things)
- How much is enough?
- How much is too much?

Insights From eXtreme Programming (XP)

- We are not endorsing XP (although many projects benefit greatly from it)
- We are not attacking RUP (although many projects have mired while using it)
- We are suggesting that knowledge about one methodology is a benefit when you use another methodlogy

Acknowledgements

- Most of these notes are a paraphrasing of the book: Extreme Programming Explained: Embrace Change by Kent Beck.
- If you are serious about learning XP, you should get a copy of this book. At a minimum, read the following:
  - Preface – Enough, pgs xviii – xix
  - Chapter 4 – Four Variables, pgs 15-19
  - Chapter 5 – Learning to Drive, pg 27
  - Chapter 10 – A Quick Overview
    - The Planning Game, pg 55
    - Refactoring, pg 58
    - 40 Hour Week, pg 60
  - Chapter 12 – Management Strategy, pgs 71-76
  - Chapter 14 – Splitting Business and Technical Responsibility, pgs 81-84
  - Chapter 15 – Planning Strategy, pgs 85-96

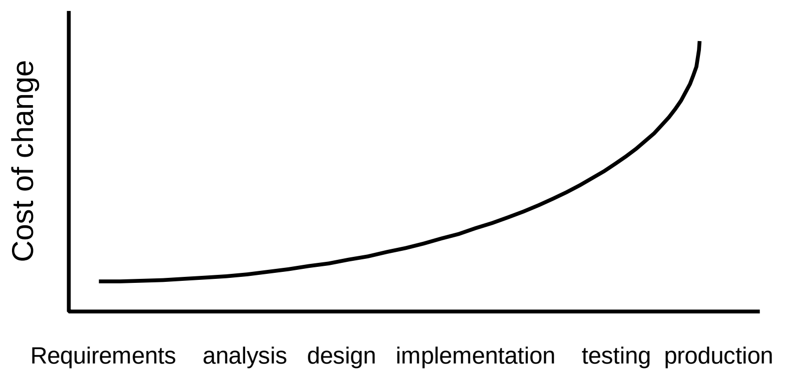
Insights from eXtreme Programming (XP)

- XP is based on the following beliefs:
  - programming can be fun
  - can't anticipate future
  - need feedback
  - must eliminate waste
  - the right roles for the right people
  - works only for small development teams
- Requires a lightweight methodology

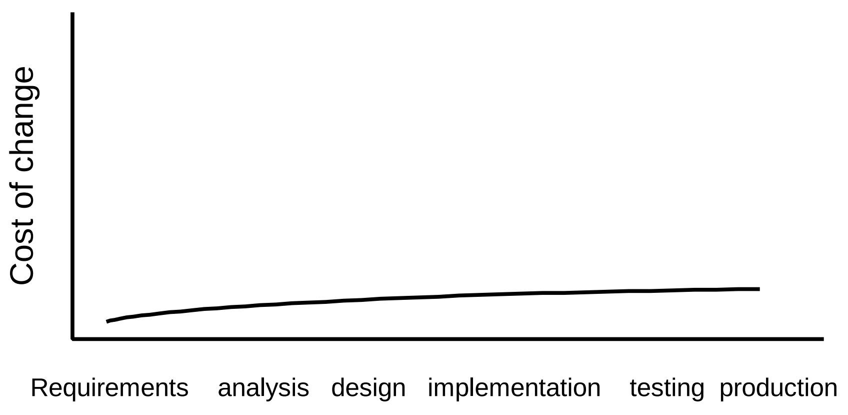
Programming Myths

- We can collect good (and complete) requirements.
- We spend a lot of time fixing bugs.
- We can anticipate change.
- We can control the important aspects of a software development project.
- Cost of change is inherently higher as the project moves forward.
- Working harder (longer) accomplishes more.
  - Manager: "OK, so I know I can't get 9 women to make a baby in a month, but if I get ***18*** women, I can do it."

Normal Thoughts About Cost of Change



Is This Possible?



What Would We Have to Do?

- Eliminate waste
- Enable change to happen quickly
- If fixing (coding) is fast, how can we eliminate bugs?

Does XP Work?

- XP *may* work.
- However, it does not scale.
  - good for teams from 2-12
- Lessons learned from XP, however, may be useful.

XP’s Four Values

- Communication
- Simplicity
- Feedback
- Courage

Four Values:
Communication

- Fraught with errors
- Need good communication - this is not the norm
- Improved with frequent contact
- Developer - developer
- Developer - customer
- Developer - business owner

Four Values:
Simplicity

- “What is the simplest that could possibly work?”

- May not need more than that.
- Wait until you need it.
- May find a better way.
- Easier to learn.
- Try it, see if it works, then correct.

Four Values:
Feedback

- We’re trying to steer our project.
- Not point it properly from the front.
- Can’t communicate as well as we’d like, so need feedback.
- Helps us correct.
- Allows us to try things and then see what we really need.
- Best way to get proper requirements.

Four Values:
Courage

- Seeing what you need to do sometimes takes courage.
- If something’s wrong - *you’ve got to fix it.*
- You may not like what reality is telling you - *but ignoring it isn’t going to work.*

- XP - see reality, have the courage to act on it.

Basic Principles

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change
- Quality work

XP Practices

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40 hour week
- on-site customer
- Coding Standards

XP Practices
Planning Game

- Responsibilities
  - business people
  - technical people
- Stories
- The process

XP Practices
The Planning Game - Responsibilities

- Business people decide:
  - – scope
  - – priority
  - – composition of releases
  - – dates of releases
- Technical people decide:
  - – estimates
  - – consequences
  - – process
  - – detailed scheduling

XP Practices
Small Releases

- gives feedback
- allows for quick corrections
- can measure progress accurately
- causes other problems (that can be solved)

XP Practices - Metaphor

- Used to guide the design.
- A potential problem -- not enough guidance
- CHALLENGE:
  – this often just isn't enough
- REFINEMENT:
  – conceptual design using UML

XP Practices - Simple Design

- Simplest is best
- May not need it in the future
- May have a better way in the future
- CHALLENGE:
  – too simple can be slower
  – should take advantage of past knowledge.
- REFINEMENT:
  – Design patterns may show us a better way now.
  – May not recover from simplistic approach.

XP Practices - Testing

- Up front testing
- You don’t know if it works until it’s tested (corollary - it doesn’t work until it’s tested)
- JUNIT works well for unit testing
- CHALLENGE:
  - doesn’t work well with GUIs
- REFINEMENT:
  - may need automated testing procedure
  - design patterns can help encapsulate change
  - MVC et. al. can help designs.

XP Practices - Refactoring

- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the changes of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written.[1]
- 1 Refactoring: Improving the Design of Existing Code, Martin Fowler. Page xvi.

Lessons Learned From XP

- Create Use-Cases to high level needs
- Detail only those “stories” used early
- Use “planning game” for project management
- Do not introduce more detail in use-cases or design unless it is useful early on
- Have more, shorter, iterations
- Require up-front, automated testing by developers
- Include users at all phases of development
- You cannot stop variation, you cannot anticipate it, but you can manage it

Bibliography

- Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helms, Johnson, Vlissides
- Extreme Programming Explained, Beck
- Java Design, Coad
- Multiparadigm Design for C++, Coplien (excellent description of commonality/variability analysis -- I even recommend the first part for Java programmers)
- Refactoring: Improving the Design of Existing Code, Fowler
- UML Distilled, Fowler
- Use Case Driven Object Modeling With Uml : A Practical Approach, Rosenberg, Scott

- Resources:
  - www.xprefined.com -- Net Objectives XP site

Net Objectives - Who We Are

- We provide training, mentoring and consulting for all phases of object-oriented software development.
- We assist companies transitioning to object-oriented development by providing mentoring throughout the entire development process.
- This enables our clients a cost-effective way to gain experience.

- **Subscribe to e-zine by sending e-mail to info@netobjectives.com**
- Contact Alan Shalloway at 425-260-8754 or at alshall@netobjectives.com

Upcoming Net Objectives Courses

- Introduction to XML (March 19, full day)
- Java and XML lab (March 20, full day)
- Introduction to OOD (April 2, half day)
- Pattern Oriented Design: Design Patterns From Analysis To Implementation (April 3-4, full day)

- Get details at:
  – http://www.netobjectives.com/pc_dps.htm