The Spirit of a Software Development Process

tim.barrios@us.ibm.com
Detailed Plans Are Not Followed

- Gant charts consist of a N tasks with some probability of completion

\[
P(\text{Executing Plan}) = \prod_{n=1}^{N} P(\text{task}_n)
\]

<table>
<thead>
<tr>
<th>N</th>
<th>P(task(_n))</th>
<th>P(Completing Plan)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>.95</td>
<td>.59</td>
</tr>
<tr>
<td>25</td>
<td>.95</td>
<td>.27</td>
</tr>
<tr>
<td>50</td>
<td>.95</td>
<td>.07</td>
</tr>
<tr>
<td>100</td>
<td>.95</td>
<td>.006</td>
</tr>
<tr>
<td>1500</td>
<td>.95</td>
<td>10(^{-23})</td>
</tr>
</tbody>
</table>
Agenda

- The Spirit of a Software Development Process - RUP
- Rational’s Internal Adoption of the RUP
- The RUP Product
- Common Pitfalls
- Summary
# Software Costs = $E \times (\text{Size})^P$

<table>
<thead>
<tr>
<th>$E$</th>
<th>Environment technologies and tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Integrated tools (compiler-editor-debugger-CM)</td>
</tr>
<tr>
<td></td>
<td>• Open systems</td>
</tr>
<tr>
<td></td>
<td>• Hardware platform performance</td>
</tr>
<tr>
<td></td>
<td>• Automation of documentation, testing, quality analyses</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{Size}$</th>
<th>(Human generated code) \ Component based development technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Reuse, commercial-off-the-shelf (COTS) products</td>
</tr>
<tr>
<td></td>
<td>• Object-oriented (analysis, design, programming)</td>
</tr>
<tr>
<td></td>
<td>• Higher level languages (C++, VB, Java, etc.)</td>
</tr>
<tr>
<td></td>
<td>• CASE tools (automatic code generation)</td>
</tr>
<tr>
<td></td>
<td>• Distributed middleware</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P$</th>
<th>Process technologies and teamwork</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Iterative development</td>
</tr>
<tr>
<td></td>
<td>• Process maturity models</td>
</tr>
<tr>
<td></td>
<td>• Architecture-first development</td>
</tr>
<tr>
<td></td>
<td>• Acquisition reform</td>
</tr>
<tr>
<td></td>
<td>• Training and personnel skill development</td>
</tr>
</tbody>
</table>
Software Development Process

- Methodology = Process + Language
- Two approaches to development
  - Idealized process (top down)
  - Patterns of success and failures
- Root cause analysis
  - Symptoms
  - Root causes
  - Best practices
Supporting Best Practices

Develop Iteratively
- Ensures users involved as requirements evolve
- Validates architectural decisions early on
- Addresses complexity of design/implementation incrementally
- Measures quality early and often
- Evolves baselines incrementally

Manage Requirements
Use Component Architectures
Model Visually
Verify Quality
Control Changes
The Spirit of The Rational Unified Process

1. Attack major risks early and continuously… or they attack you
2. Ensure that you deliver value to your customer
3. Have a maniacal focus on working software
4. Accommodate change early in the project
5. Baseline an executable architecture early on
6. Build your system with components
7. Work closely together as one team
8. Make quality a way of life, not an afterthought
Waterfall Development

- Late discovery of issues
- Subjective and error-prone measure of progress
- Late integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations
Risk Profile: Waterfall Projects
What Happens in Practice

Sequential activities:

Requirements ➔ Design ➔ Code ➔ Integration ➔ Test
Waterfall: Hard to Scale up

- A waterfall approach can not properly handle the growing complexity associated with
  - Increased duration
  - Increased application size
  - Larger and/or distributed team
  - Increased technical complexity
  - Novelty of technology

- The root cause of the problem with the waterfall lifecycle is that it does not allow to identify and mitigate risks early enough
Flavors of Iterative Development

- Spiral Model
  - Barry Boehm, 1987
- Incremental delivery
  - Tom Gilb, 1988
- Dynamic Systems Development Method
  - Jennifer Stapleton
- Unified Process
  - Kruchten, Jacobson, Booch, Rumbaugh
Iterative Development

- Earliest iterations address greatest risks
- Each iteration produces an executable release
- Each iteration includes integration and test
## Iterative Development Phases

### Major Milestones

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

### Time

- **Inception:** Define the scope of project
- **Elaboration:** Plan project, specify features, baseline architecture
- **Construction:** Build the product
- **Transition:** Transition the product into end-user community
An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release.
Risk Profile: *Waterfall vs. Iterative Projects*
Iterative Development Accelerates Risk Reduction

Diagram showing the comparison between Iterative Development and Waterfall methods, highlighting reduction in risk over time with iterative approaches.
Better Progress Profile
Sequential phases, but iterative activities

Prototypes ➔ Architecture ➔ Functional Releases ➔ Product Release

Modern Project Profile

Waterfall Project Profile

Development Progress (% Coded)

Project Schedule
Risk Mitigation: Hitting Hard Problems Earlier

- When the initial risks are mitigated, new ones emerge
- Do not do just the easy stuff, to look good
- Keep re-planning based on all new information
- In iterative development, you cannot lie to yourself very long
2. Ensure That You Deliver Value to Your Customer

- Focus on key requirements
  - Capture, document
  - Organize, prioritize
- Requirements will change
  - Evaluate impact of change and decide what changes to implement
  - Propagate changes to all team members
- Make requirements accessible

Requirements management leverages your ability to deliver products that meet user needs
Use-Case Driven Development

- A use case describes complete and meaningful services that your system offers to users and other systems
- Use cases drive the work through each iteration
  - Planning of iterations
  - Creation and validation of the architecture
  - Definition of test cases and procedures
  - Design of user interfaces and creation of user documentation
3. Have a Maniacal Focus on Working Software

- Measure progress primarily by reviewing executable code, and test results
  - Plans, requirements, designs and other by-products often provide a false perception of progress and status
- Focus on the final, delivered product, and only the artifacts that matter to get at this goal consistently
  - Streamline the process
  - Do not use all of the RUP! Only use what makes sense to your project
4. Accommodate Change Early in the Project

Today’s systems are too complex to get the requirements, architecture, design, implementation and scope right the first time.

Provides freedom to change:

- Business Solution
- Architecture
- Design & Implementation
- Scope (Reduction)

- Inception
  - Preliminary Iteration
- Elaboration
  - Architect. Iteration
- Construction
  - Devel. Iteration
- Transition
  - Transition Iteration
Cost of Change

The graph illustrates the cost of change over different iterations of a project. The cost increases significantly as the project progresses through various phases, such as Inception, Elaboration, Construction, and Transition. The graph shows different categories of cost, including Business Solution, Architecture, Design & Implementation, and Scope Reduction, each represented by different colors and shaded areas.
5. Baseline an Executable Architecture Early

- Architecture provides a skeleton structure of your system
  - Subsystems, key components, interfaces, architectural mechanisms (solutions for common problems, such as persistency, inter-process communication, …)
- Implementing and testing the architecture mitigates most technical risks

**Produce Executable Architecture**

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Devel. Iter.</td>
<td>Transition Iter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transition Iter.</td>
</tr>
</tbody>
</table>
The Spirit of RUP

- Attack major risks early and continuously… or they attack you
- Ensure that you deliver value to your customer
- Have a maniacal focus on working software
- Accommodate change early in the project
- Baseline an executable architecture early on
- Build your system with components
- Work closely together as one team
- Make quality a way of life, not an afterthought
Traditional Functional Decomposition

- Requirements driven
- Many dependencies creates inflexible systems
Architecture-driven, Component-based Design

- Component architecture provides flexibility
Example: **Component-Based Architecture**

- E-commerce Functions
  - Customer Data
  - Product Data

- Personalization Functions
  - Customer Personalization
  - Product Personalization

- User Interface Mechanisms
  - Purchased
  - Existing
  - New
7. Work Closely Together As One Team

- Empowered and self-managed
  - Clear vision
- Accountable for team results
  - Clear expectations
  - All for one, one for all - avoid “My design was good, your code didn’t work”
- Optimized communication
  - Face-to-face rather than e-mail
  - Effective process (right-sized for your project)
  - Organize around architecture, not around functions
  - Get the right tool support
    - Easy access to current requirement
    - Private workspaces
    - Easy access to defects….
    - …
8. Make Quality a Way of Life, Not an Afterthought

Software problems are 100 to 1000 times more costly to find and repair after deployment

- Cost to Repair Software
- Cost of Lost Opportunities
- Cost of Lost Customers

Cost

Inception | Elaboration | Construction | Transition
Test Each Iteration

UML Model and Implementation

Tests

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Test Suite 1

Test Suite 2

Test Suite 3

Test Suite 4
Agenda

- The Spirit of the RUP
- Rational’s Internal Adoption of the RUP
- The RUP Product
- Common Pitfalls
- Summary
Rational Development Locations: *Teams of Teams*
Rational Suite Development Team

- +700 engineers
- 3 continents, 11 sites
- 15 integrated products
- 2 synchronized releases per year
  - 6 releases so far
- Tested on 9 languages on 7 platforms
  - That means 63 combinations to test!
  - And growing…
- Internationalization and localization
Rational’s Internal Process

- Project length: 12 months
  - 9-10 iterations a 4-6 weeks
- Weekly and then daily builds the last +100 days
  - Each build is complete with installer. Tested on multiple platforms
  - Possible due to CM and automated test tools
- Common requirements and defect databases
  - Possible through tool automation
Rational’s Internal Process

- Metrics collected and distributed daily
  - Closely monitors trends in defects and churn in requirements and code
  - Possible due to project monitoring tools
- Visual modeling and component-based development
  - Possible through modeling tools
- Change Control Board
  - After certain date, all changes need formal approval
- Common understanding of how we develop software
  - Possible through a common process
Agenda

- The Spirit of the RUP
- Rational’s Internal Adoption of the RUP
- The RUP Product
- Common Pitfalls
- Summary
Accessing the RUP

- Interactive knowledge base accessible from tools
- Web-enabled with powerful graphical navigation
- Search engine and index
Configuring the RUP

- Right-size your process with RUP Builder

**Partners**
- Technology Plug-In
- Tool Plug-In
- Domain Plug-In

**Core RUP**
- Core Plug-In
- Core Plug-In
- Core Plug-In
- Core Plug-In

**Customers**
- Project Plug-In
- Company Plug-In

**RUP Configurations**
- RUP Realtime
- Small RUP
- RUP for .NET
- RUP for J2EE
- RUP for CGEB
- XYZ
Agenda

- The Spirit of the RUP
- Rational’s Internal Adoption of the RUP
- The RUP Product
- Common Pitfalls
- Summary
Common Pitfalls in Inception

- **Too much formality / too many artifacts**
  - Only produce the artifacts that add value, minimize formality if possible
  - When in doubt of value, don’t do it

- **Analysis Paralysis**
  - You can improve upon things later on – move on
  - Focus on objectives with Inception
  - Do NOT describe all requirements in detail

- **Too long initial iteration**
  - Cut scope rapidly
  - You fail with first iteration, project likely to fail
Common Pitfalls in Elaboration

- Functional, Specialized Organization
  - Teams of generalists and multitasking experts
  - No place for “I only do <X>” mentality
- Save the tricky part for later
  - Attack risks early, or they attack you
  - Hard on you now, but makes life easier later
- No implementation and validation of architecture
  - You cannot get the architecture right or address major risks without implementing and testing the architecture
- No willingness to change things
  - Change enables improvement
Common Pitfalls in Construction

- Basing work on unstable architecture
  - Major rework and integration issues
  - High price to pay for insufficient work in Elaboration
- Reinventing solutions to common problems
  - Were architectural mechanisms (patterns) developed in Elaboration and communicated to everybody?
- Continuous integration not happening
  - Daily or weekly build minimizes rework
- Testing not initiated until end of construction
  - You are unlikely to meet deadlines
  - Beta may be of too low quality to offer value
Common Pitfalls in Transition

- Not enough beta users
  - Did you have beta customers lined up and prepared at end of Construction?
- Not all functionality beta tested
  - Did you include the functionality in the beta release?
  - Was it usable? (Ease of use, performance, documented, …)
- Customer not happy with delivered functionality
  - Was acceptance criteria approved by customer?
  - Did you involve customer throughout the project?
Agenda

- The Spirit of the RUP
- Rational’s Internal Adoption of the RUP
- The RUP Product
- Common Pitfalls
- Summary
Summary

- The Spirit of the Rational Unified Process
  - Attack major risks early and continuously...or they attack you
  - Ensure that you deliver value to your customer
  - Have a maniacal focus on working software
  - Accommodate change early in the project
  - Baseline an executable architecture early on
  - Build your system with components
  - Work closely together as one team
  - Make quality a way of life, not an afterthought
Further Reading and Exploration

- Walker Royce, Software project management—A Unified Framework, Addison-Wesley (1998)