Software Engineering
G22.2440-001

Session 1 - Main Theme
Software Engineering Fundamentals

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

- Course Overview and Logistics
- Software Engineering Scope
- Software Engineering Discipline
- Software Development Challenges
- Refining the Software Engineering Discipline
- The Human Side of Software Development
- Software Engineering Best Practices ala Rational
- Rational Unified Process (RUP)
- Introduction to Agile Software Engineering
- Summary
  - Course Assignments
  - Course Project
  - Readings
Session Objectives

- Define software engineering and explain its importance
- Discuss the concepts of software products and software processes
- Explain the importance of process visibility
- Introduce the notion of professional responsibility
- Introduce Agile Software Engineering

Part I

Course Overview and Logistics
Course Logistics

- Course Web Site
  - [http://www.nyu.edu/classes/jcf/g22.2440-001](http://www.nyu.edu/classes/jcf/g22.2440-001)
  - Login/Password (not currently protected)
  - Review syllabus

- Textbooks
  - Software Engineering: A Practitioner's Approach with Bonus Chapter on Agile Development
    Roger S. Pressman
    McGraw-Hill Science/Engineering/Math

Useful Knowledge

- Business Process Modeling (BPML)
- Object-Oriented Analysis and Design (OOAD)
- Object-oriented technology experience
- Software development experience as a software development team member in the role of business analyst, developer, or project manager
- Implementation language experience (e.g., C++, Java)
- Note: Knowledge of UML or a specific programming language is not required
## Course Objectives

- Present modern software engineering techniques and examines the software life-cycle, including software specification, design implementation, testing and maintenance.
- Describe and compare various software development methods and understand the context in which each approach might be applicable.
- Develop students’ critical skills to distinguish sound development practices from ad hoc practices, judge which technique would be most appropriate for solving large-scale software problems, and articulate the benefits of applying sound practices.

## Course Objectives (continued)

- Expand students’ familiarity with mainstream languages used to model and analyze processes and object designs (e.g., BPML, UML).
- Demonstrate the importance of formal/executable specifications of object models, and the ability to verify the correctness/completeness of solution by executing the models.
- Explain the scope of the software maintenance problem and demonstrate the use of several tools for reverse engineering software.
Course Objectives (continued)

- Develop students’ ability to evaluate the effectiveness of an organization’s SW development practices, suggest improvements, and define a process improvement strategy
- Introduce state-of-the-art tools and techniques for large-scale development
- Implement the major software development methods in practical projects and motivate discussion via group presentations

Software Requirements

- Microsoft Windows 2000 / 2003 / XP Professional
- Software tools will be available from the Internet or from the course Web site under demos as a choice of freeware or commercial tools
  - Business and Application Modeling Tools
  - Software Development Tools
  - Workflow Management Frameworks
  - etc.
- References will be provided on the course Web site
Part II

Software Engineering Scope

Software Engineering

- The economies of ALL developed nations are dependent on software
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development
- Software engineering expenditure represents a significant fraction of GNP in all developed countries
  - GNP stands for Gross National Product. GNP per capita is the dollar value of a country’s final output of goods and services in a year, divided by its population. It reflects the average income of a country’s citizens.
Software Costs

- Software costs often dominate system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Software Products

- Generic products
  - Stand-alone systems which are produced by a development organization and sold on the open market to any customer.
- Bespoke (customized) products
  - Systems which are commissioned by a specific customer and developed specially by some contractor.
- Most software expenditure is on generic products but most development effort is on bespoke systems.
Software Product Attributes

- **Maintainability**
  - It should be possible for the software to evolve to meet changing requirements

- **Dependability**
  - The software should not cause physical or economic damage in the event of failure

- **Efficiency**
  - The software should not make wasteful use of system resources

- **Usability**
  - Software should have an appropriate user interface and documentation

Importance of Product Characteristics

- The relative importance of these characteristics depends on the product and the environment in which it is to be used

- In some cases, some attributes may dominate
  - In safety-critical real-time systems, key attributes may be dependability and efficiency

- Costs tend to rise exponentially if very high levels of any one attribute are required
Software engineering is concerned with the theories, methods and tools for developing, managing and evolving software products. Software products consist of programs and documentation. Product attributes are maintainability, dependability, efficiency and usability.
Part III

*Software Engineering Discipline*

The Software Process

- Structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution
- Activities vary depending on the organization and the type of system being developed
- Must be explicitly modeled if it is to be managed
Process Characteristics

- Understandability
  - Is the process defined and understandable?

- Visibility
  - Is the process progress externally visible?

- Supportability
  - Can the process be supported by CASE tools?

- Acceptability
  - Is the process acceptable to those involved in it?

Process Characteristics

- Reliability
  - Are process errors discovered before they result in product errors?

- Robustness
  - Can the process continue in spite of unexpected problems?

- Maintainability
  - Can the process evolve to meet changing organizational needs?

- Rapidity
  - How fast can the system be produced?
Engineering Process Model

- Specification - set out the requirements and constraints on the system
- Design - Produce a paper model of the system
- Manufacture - build the system
- Test - check if the system meets the required specifications
- Install - deliver the system to the customer and ensure it is operational
- Maintain - repair faults in the system as they are discovered

Software Process Models

- Normally, specifications are incomplete/anomalous
- Very blurred distinction between specification, design and manufacturing
- No physical realization of the system for testing
- Software does not wear out - maintenance does not mean component replacement
Generic Software Process Models

- The waterfall model
  - Separate and distinct phases of specification and development
- Evolutionary development
  - Specification and development are interleaved
- Formal transformation
  - A mathematical system model is formally transformed to an implementation
- Reuse-based development
  - The system is assembled from existing components

Waterfall Model
Waterfall Model Phases

- Phases:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

Evolutionary Development

Concurrent activities

- Specification
- Development
- Validation

Outline description

Initial version
Intermediate versions
Final version
Evolutionary Development

- Exploratory prototyping
  - Objective is to work with customers and to evolve a final system from an initial outline specification
  - Should start with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the system requirements
  - Should start with poorly understood requirements

Problem

- Lack of process visibility
- Systems are often poorly structured
- Special skills (e.g., languages for rapid prototyping) may be required

Applicability

- For small or medium-size interactive systems
- For parts of large systems (e.g. the user interface)
- For short-lifetime systems
Key Points

- The software process consists of those activities involved in software development
- The waterfall model considers each process activity as a discrete phase
- Evolutionary development considers process activities as concurrent

Part IV

Software Development Challenges
Inherent Risks


- Sponsorship
- Budget
- Culture
- Business Understanding
- Priorities
  - Business changes
  - Features
  - Schedule slips
- Methodology Misuse
- Software Quality

Symptoms of Software Development Problems

- User or business needs not met
- Requirements churn
- Modules don’t integrate
- Hard to maintain
- Late discovery of flaws
- Poor quality of end-user experience
- Poor performance under load
- No coordinated team effort
- Build-and-release issues
Trace Symptoms to Root Causes

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Root Causes</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs not met</td>
<td>Insufficient requirements</td>
<td>Develop Iteratively</td>
</tr>
<tr>
<td>Requirements churn</td>
<td>Ambiguous communications</td>
<td></td>
</tr>
<tr>
<td>Modules don't fit</td>
<td>Brittle architectures</td>
<td>Manage Requirements</td>
</tr>
<tr>
<td>Hard to maintain</td>
<td>Overwhelming complexity</td>
<td>Use Component Architectures</td>
</tr>
<tr>
<td>Late discovery</td>
<td>Undetected inconsistencies</td>
<td></td>
</tr>
<tr>
<td>Poor quality</td>
<td>Poor testing</td>
<td></td>
</tr>
<tr>
<td>Poor performance</td>
<td>Subjective assessment</td>
<td></td>
</tr>
<tr>
<td>Colliding developers</td>
<td>Waterfall development</td>
<td>Model Visually (UML)</td>
</tr>
<tr>
<td>Build-and-release</td>
<td>Uncontrolled change</td>
<td>Continuously Verify Quality</td>
</tr>
<tr>
<td></td>
<td>Insufficient automation</td>
<td>Manage Change</td>
</tr>
</tbody>
</table>

Risk Management

- Perhaps the principal task of a manager is to minimize risk
- The 'risk' inherent in an activity is a measure of the uncertainty of the outcome of that activity
- High-risk activities cause schedule and cost overruns
- Risk is related to the amount and quality of available information. The less information, the higher the risk
Process Model Risk Problems

- Waterfall
  - High risk for new systems because of specification and design problems
  - Low risk for well-understood developments using familiar technology
- Prototyping
  - Low risk for new applications because specification and program stay in step
  - High risk because of lack of process visibility
- Transformational
  - High risk because of need for advanced technology and staff skills

Part V

*Refining the Software Engineering Discipline*
Hybrid Process Models

- Large systems are usually made up of several sub-systems
- The same process model need not be used for all subsystems
- Prototyping for high-risk specifications
- Waterfall model for well-understood developments

Spiral Model of the Software Process
Phases of the Spiral Model

- Objective setting
  - Specific objectives for the project phase are identified

- Risk assessment and reduction
  - Key risks are identified, analyzed and information is sought to reduce these risks

- Development and validation
  - An appropriate model is chosen for the next phase of development.

- Planning
  - The project is reviewed and plans drawn up for the next round of the spiral

Template for a Spiral Round

- Objectives
- Constraints
- Alternatives
- Risks
- Risk resolution
- Results
- Plans
- Commitment
Quality Improvement

- Objectives
  - Significantly improve software quality

- Constraints
  - Within a three-year timescale
  - Without large-scale capital investment
  - Without radical change to company standards

- Alternatives
  - Reuse existing certified software
  - Introduce formal specification and verification
  - Invest in testing and validation tools

Risk

- Risks
  - No cost effective quality improvement
  - Possible quality improvements may increase costs excessively
  - New methods might cause existing staff to leave

- Risk resolution
  - Literature survey
  - Pilot project
  - Survey of potential reusable components
  - Assessment of available tool support
  - Staff training and motivation seminars
Approach

- Results
  - Experience of formal methods is limited - very hard to quantify improvements
  - Limited tool support available for company-wide standard development system
  - Reusable components available but little support exists in terms of reusability tools

- Plans
  - Explore reuse option in more detail
  - Develop prototype reuse support tools
  - Explore component certification scheme

- Commitment
  - Fund further 18-month study phase

Catalogue Spiral

- Objectives
  - Procure software component catalogue

- Constraints
  - Within a year
    - Must support existing component types
    - Total cost less than $100, 000

- Alternatives
  - Buy existing information retrieval software
  - Buy database and develop catalogue using database
  - Develop special purpose catalogue
**Risk**

- **Risks**
  - May be impossible to procure within constraints
  - Catalogue functionality may be inappropriate

- **Risk resolution**
  - Develop prototype catalogue (using existing 4GL and an existing DBMS) to clarify requirements
  - Commission consultants report on existing information retrieval system capabilities.
  - Relax time constraint

---

**Approach**

- **Results**
  - Information retrieval systems are inflexible.
  - Identified requirements cannot be met.
  - Prototype using DBMS may be enhanced to complete system
  - Special purpose catalogue development is not cost-effective

- **Plans**
  - Develop catalogue using existing DBMS by enhancing prototype and improving user interface

- **Commitment**
  - Fund further 12 month development
Spiral Model Flexibility

- Hybrid models accommodated for different parts of a project:
  - Well-understood systems (low technical risk)
    - Use Waterfall model as risk analysis phase is relatively cheap
  - Stable requirements and formal specification with safety criticality
    - Use Formal transformation model
  - High UI risk with incomplete specification
    - Use Prototyping model

Spiral Model Advantages

- Focuses attention on reuse options
- Focuses attention on early error elimination
- Puts quality objectives up front
- Integrates development and maintenance
- Provides a framework for hardware/software development
Spiral Model Problems

- Contractual development often specifies process model and deliverables in advance
- Requires risk assessment expertise
- Needs refinement for general use

Process Visibility

- Software systems are intangible so managers need documents to assess progress
- However, this may cause problems
  - Timing of progress deliverables may not match the time needed to complete an activity
  - The need to produce documents constraints process iteration
  - The time taken to review and approve documents is significant
- Waterfall model is still the most widely used deliverable-based model
### Waterfall Model Documents

<table>
<thead>
<tr>
<th>Activity</th>
<th>Output documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements analysis</td>
<td>Feasibility study, Outline requirements</td>
</tr>
<tr>
<td>Requirements definition</td>
<td>Requirements document</td>
</tr>
<tr>
<td>System specification</td>
<td>Functional specification, Acceptance test plan</td>
</tr>
<tr>
<td></td>
<td>Draft user manual</td>
</tr>
<tr>
<td>Architectural design</td>
<td>Architectural specification, System test plan</td>
</tr>
<tr>
<td>Interface design</td>
<td>Interface specification, Integration test plan</td>
</tr>
<tr>
<td>Detailed design</td>
<td>Design specification, Unit test plan</td>
</tr>
<tr>
<td>Coding</td>
<td>Program code</td>
</tr>
<tr>
<td>Unit testing</td>
<td>Unit test report</td>
</tr>
<tr>
<td>Module testing</td>
<td>Module test report</td>
</tr>
<tr>
<td>Integration testing</td>
<td>Integration test report, Final user manual</td>
</tr>
<tr>
<td>System testing</td>
<td>System test report</td>
</tr>
<tr>
<td>Acceptance testing</td>
<td>Final system plus documentation</td>
</tr>
</tbody>
</table>

### Process Model Visibility

<table>
<thead>
<tr>
<th>Process model</th>
<th>Process visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall model</td>
<td>Good visibility, each activity produces some deliverable</td>
</tr>
<tr>
<td>Evolutionary development</td>
<td>Poor visibility, uneconomic to produce documents during rapid iteration</td>
</tr>
<tr>
<td>Formal transformations</td>
<td>Good visibility, documents must be produced from each phase for the process to continue</td>
</tr>
<tr>
<td>Reuse-oriented development</td>
<td>Moderate visibility, it may be artificial to produce documents describing reuse and reusable components.</td>
</tr>
<tr>
<td>Spiral model</td>
<td>Good visibility, each segment and each ring of the spiral should produce some document.</td>
</tr>
</tbody>
</table>
Key Points

- The spiral process model is risk-driven
- Process visibility involves the creation of deliverables from activities

Part VI

The Human Side of Software Development
Professional Responsibility

- Software engineers should not just be concerned with technical considerations. They have wider ethical, social and professional responsibilities.
- Not clear what is right or wrong about the following issues:
  - Development of military systems
  - Whistle blowing
  - What is best for the software engineering profession

Ethical Issues

- Confidentiality
- Competence
- Intellectual property rights
- Computer misuse
Key Points

- Software engineers have ethical, social and professional responsibilities

Part VII

*Software Engineering Best Practices ala Rational*
Section Outline

- Identify Steps for Understanding and Solving Software Engineering Problems
- Explain the Rational Six Best Practices
- Present the Rational Unified Process within the context of the Six Best Practices

Practice 1: Develop Iteratively

<table>
<thead>
<tr>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Made Practical</td>
</tr>
<tr>
<td><strong>Develop Iteratively</strong></td>
</tr>
<tr>
<td>Manage Requirements</td>
</tr>
<tr>
<td>Use Component Architectures</td>
</tr>
<tr>
<td>Model Visually (UML)</td>
</tr>
<tr>
<td>Continuously Verify Quality</td>
</tr>
<tr>
<td>Manage Change</td>
</tr>
</tbody>
</table>
Waterfall Development Characteristics

- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

Iterative Development Produces Executable Releases

Each iteration results in an executable release.
Practice 2: Manage Requirements

Best Practices
*Process Made Practical*

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change
Requirements Management

- Making sure you
  - Solve the right problem
  - Build the right system
- By taking a systematic approach to
  - eliciting
  - organizing
  - documenting
  - managing
  the changing requirements of a software application.

Aspects of Requirements Management

- Analyze the Problem
- Understand User Needs
- Define the System
- Manage Scope
- Refine the System Definition
- Build the Right System
Practice 3: Use Component Architectures

Best Practices
Process Made Practical

- Develop Iteratively
- Manage Requirements
- **Use Component Architectures**
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change
Resilient, Component-Based Architectures

- Resilient
  - Meets current and future requirements
  - Improves extensibility
  - Enables reuse
  - Encapsulates system dependencies

- Component-based
  - Reuse or customize components
  - Select from commercially-available components
  - Evolve existing software incrementally

Purpose of a Component-Based Architecture

- Basis for reuse
  - Component reuse
  - Architecture reuse

- Basis for project management
  - Planning
  - Staffing
  - Delivery

- Intellectual control
  - Manage complexity
  - Maintain integrity
Practice 4: Model Visually (UML)

**Best Practices**
*Process Made Practical*

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- **Model Visually (UML)**
- Continuously Verify Quality
- Manage Change

---

**Why Model Visually?**

- Capture structure and behavior
- Show how system elements fit together
- Keep design and implementation consistent
- Hide or expose details as appropriate
- Promote unambiguous communication
  - UML: one language for all practitioners
Visual Modeling with UML

- Multiple views
- Precise syntax and semantics

**Static Diagrams**

- Use-Case Diagrams
- Class Diagrams
- Object Diagrams
- Component Diagrams

**Dynamic Diagrams**

- Sequence Diagrams
- Collaboration Diagrams
- Statechart Diagrams
- Activity Diagrams
- Deployment Diagrams

Visual Modeling Using UML Diagrams

- Use-case diagram
- Class diagram
- Statechart diagram
- Deployment diagram
- Component diagram
- Collaboration diagram
- Sequence diagram

Forward and Reverse Engineering

Target System
UML Notation Baseline

<table>
<thead>
<tr>
<th>Diagram Name</th>
<th>Type</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case</td>
<td>Static*</td>
<td>Analysis</td>
</tr>
<tr>
<td>Class</td>
<td>Static</td>
<td>Analysis</td>
</tr>
<tr>
<td>Activity</td>
<td>Dynamic**</td>
<td>Analysis</td>
</tr>
<tr>
<td>State-Transition</td>
<td>Dynamic</td>
<td>Analysis</td>
</tr>
<tr>
<td>Event Trace (Interaction)</td>
<td>Dynamic</td>
<td>Design</td>
</tr>
<tr>
<td>Sequence</td>
<td>Dynamic</td>
<td>Design</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Dynamic</td>
<td>Design</td>
</tr>
<tr>
<td>Package</td>
<td>Static</td>
<td>Delivery</td>
</tr>
<tr>
<td>Deployment</td>
<td>Dynamic</td>
<td>Delivery</td>
</tr>
</tbody>
</table>

*Static describes structural system properties
**Dynamic describes behavioral system properties.

UML Diagrams

UML defines twelve types of diagrams, divided into three categories:

- Four diagram types represent static application structure:
  - Class Diagram
  - Object Diagram
  - Component Diagram
  - Deployment Diagram

- Five represent different aspects of dynamic behavior:
  - Use Case Diagram
  - Sequence Diagram
  - Activity Diagram
  - Collaboration Diagram
  - Statechart Diagram

- Three represent ways you can organize and manage your application modules:
  - Packages
  - Subsystems
  - Models

Source: [http://www.omg.org/gettingstarted/what_is_uml.htm](http://www.omg.org/gettingstarted/what_is_uml.htm)
UML Views

- **Approach**
  - UML defines five views that let you look at overall models from various angles
  - Layering architectural principles is used to allocate pieces of functionality to subsystems
  - Partitioning is used to group related pieces of functionality into packages within subsystems

- **Views and Related Diagrams**
  - Use Case View (application functionality)
    - Use Case Diagram
  - Logical View (static application structure)
    - Class Diagram
    - Object Diagram
  - Process View (dynamic application behavior)
    - Sequence Diagram
    - Activity Diagram
    - Collaboration Diagram
    - Statechart Diagram
  - Implementation View (application packaging)
    - Component Diagram
  - Deployment View (application delivery)
    - Deployment Diagram

---

Need to Maintain Consistency and Coverage Across UML Views

![Diagram showing the relationship between different UML views](image)
Practice 5: Continuously Verify Quality

Best Practices
Process Made Practical

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change

Continuously Verify Software Quality

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 All Dimensions of Software Quality

- **Functionality**
  - Verification of each usage scenario
- **Reliability**
  - Verification of sustained application operation
  - Does the system perform under production load?
- **Performance**
  - Test performance under expected & worst-case load
  - Does my application respond acceptably?
  - Does my application do what's required?

Test Each Iteration

- **UML Model and Implementation**
  - Iteration 1
  - Iteration 2
  - Iteration 3
  - Iteration 4
- **Tests**
  - Test Suite 1
  - Test Suite 2
  - Test Suite 3
  - Test Suite 4
Practice 6: Manage Change

Best Practices
Process Made Practical

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change

What Do You Want to Control?

- Changes to enable iterative development
- Secure workspaces for each developer
- Automated integration/build management
- Parallel development

CM is more than just check-in and check-out
Aspects of a CM System

- Change Request Management
- Configuration Status Reporting
- Configuration Management (CM)
- Change Tracking
- Version Selection
- Software Manufacture

Unified Change Management

- Management across the lifecycle
  - System
  - Project Management
- Activity-Based Management
  - Tasks
  - Defects
  - Enhancements
- Progress Tracking
  - Charts
  - Reports
### Best Practices Reinforce Each Other

<table>
<thead>
<tr>
<th>Best Practices</th>
<th>Reinforces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop Iteratively</td>
<td>Ensures users involved as requirements evolve</td>
</tr>
<tr>
<td>Manage Requirements</td>
<td>Validates architectural decisions early on</td>
</tr>
<tr>
<td>Use Component Architectures</td>
<td>Addresses complexity of design/implementation incrementally</td>
</tr>
<tr>
<td>Model Visually (UML)</td>
<td>Measures quality early and often</td>
</tr>
<tr>
<td>Continuously Verify Quality</td>
<td>Evolves baselines incrementally</td>
</tr>
<tr>
<td>Manage Change</td>
<td></td>
</tr>
</tbody>
</table>

---

### Part VIII

**Rational Unified Process**

---
Foundations of RUP

- Implement Software Engineering Best Practices:
  - Iterative Controlled Development
  - Use Case Models for Business Requirements
  - Component Architectures
  - Risk Identification, Management & Mitigation

RUP Best Practices
Implementation

Best Practices
Process Made Practical
- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change
Achieving Best Practices

- Iterative Approach
- Guidance for activities and work products (artifacts)
- Process focus on architecture
- Use cases which drive design and implementation
- Models which abstract the system

A Team-Based Definition of Process

- A process defines **Who** is doing **What**, **When** and **How** to reach a certain goal.
The Rational Unified Process has four Phases:
- **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 based on an established plan and evaluation criteria, resulting in an executable release (internal or external).

Workflows Produce Models
Bringing It All Together: The Iterative Approach

Workflows group activities logically

In an iteration, you walk through all workflows

Workflows Guide Iterative Development

Business Modeling: Workflow Details

Requirements: Workflow Details
Roles Are Used for Resource Planning

<table>
<thead>
<tr>
<th>Resource</th>
<th>Role</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul</td>
<td>Designer</td>
<td>Define Operations</td>
</tr>
<tr>
<td>Mary</td>
<td>Requirements Specifier</td>
<td>Detail a Use Case</td>
</tr>
<tr>
<td>Joe</td>
<td>System Analyst</td>
<td>Find Actors and Use Cases</td>
</tr>
<tr>
<td>Sylvia</td>
<td>Implementer</td>
<td>Perform Unit Tests</td>
</tr>
<tr>
<td>Stefan</td>
<td>Architect</td>
<td>Identify Design Mechanisms</td>
</tr>
</tbody>
</table>

Each individual in the project is assigned to one or several roles.
Roles Perform Activities and Produce Artifacts

Example
Requirements: Workflow Detail
"Define the System"

Overview of Rational Unified Process Concepts
Summary: Best Practices of Software Engineering

- Best Practices guide software engineering by addressing root causes
- Best Practices reinforce each other
- Process guides a team on what to do, how to do it, and when to do it
- The Rational Unified Process is a means of achieving Best Practices

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Investment Concept Statement Business Case</td>
<td>Outlines the project’s purpose, scope, costs, benefits and risks of the investment and is used by business sponsors and stakeholders to make an informed decision</td>
</tr>
<tr>
<td>Vision</td>
<td>Defines the stakeholders view of the product to be developed, contains an outline of the envisioned core requirements, defines the boundary and primary features of the system and is used as the basis for more detailed technical requirements</td>
</tr>
<tr>
<td>Stakeholders Requests</td>
<td>Captures all requests made on the project from stakeholders</td>
</tr>
<tr>
<td>Technology Governance Questionnaire</td>
<td>Assesses the impact of all development projects introducing significant architectural or high-level design changes</td>
</tr>
<tr>
<td>Use Case Specifications</td>
<td>Defines the functional requirements for the system with use case diagrams</td>
</tr>
<tr>
<td>Supplementary Specifications</td>
<td>Defines the nonfunctional requirements of the system</td>
</tr>
<tr>
<td>Software Architecture Document</td>
<td>Provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system - use case view, logical view, process view, deployment view, implementation view and data view (as needed)</td>
</tr>
<tr>
<td>User Acceptance Test Plan</td>
<td>Documents a plan to be used to direct user acceptance testing and ensures that all of the detailed business requirements defined in Inception are tested completely</td>
</tr>
<tr>
<td>System Test Plan</td>
<td>Outlines and communicates the objectives of the testing effort to gain acceptance and approval from the stakeholders</td>
</tr>
<tr>
<td>Corporate Report Card</td>
<td>Provides measurement and explanation of variances between actual and expected project performance and informs management of project issues (High Risk, High Impact)</td>
</tr>
<tr>
<td>Issues List</td>
<td>Entails the documentation, review, resolution, and follow-up of project issues</td>
</tr>
<tr>
<td>Risk List</td>
<td>Details a list of known and open risks to the project, sorted in decreasing order of importance and associated with specific mitigation strategies or contingency plans</td>
</tr>
<tr>
<td>Project Plan / Iteration Plan</td>
<td>Details the specific tasks that must be completed by each team member in order to complete a project</td>
</tr>
<tr>
<td>Phase Assessment Review</td>
<td>Establishes criteria for determining whether or not a project is ready to move from one phase to the next phase</td>
</tr>
</tbody>
</table>
RUP Core Artifacts

<table>
<thead>
<tr>
<th>Phase</th>
<th>S</th>
<th>M</th>
<th>L</th>
<th>Artifact</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td></td>
<td></td>
<td></td>
<td>Investment Concept Statement</td>
<td>Business Sponsor (A)</td>
</tr>
<tr>
<td>Inception</td>
<td></td>
<td></td>
<td></td>
<td>Business Case</td>
<td>Business Project Manager (A)</td>
</tr>
<tr>
<td>Inception</td>
<td>Light</td>
<td></td>
<td></td>
<td>Vision</td>
<td>Business Lead (A)</td>
</tr>
<tr>
<td>Inception</td>
<td>Vision</td>
<td></td>
<td></td>
<td>Stakeholder Requests</td>
<td>Business Lead</td>
</tr>
<tr>
<td>Inception</td>
<td></td>
<td></td>
<td></td>
<td>Delegated Governance Questionnaire</td>
<td>Technology Project Manager</td>
</tr>
<tr>
<td>Elaboration</td>
<td></td>
<td></td>
<td></td>
<td>Use Case Specifications</td>
<td>Business Lead (A)</td>
</tr>
<tr>
<td>Elaboration</td>
<td>Vision</td>
<td></td>
<td></td>
<td>Supplementary Specifications</td>
<td>Technology Project Manager (A)</td>
</tr>
<tr>
<td>Elaboration</td>
<td></td>
<td></td>
<td></td>
<td>Software Architecture Document</td>
<td>Technology Project Manager</td>
</tr>
<tr>
<td>Construction</td>
<td></td>
<td></td>
<td></td>
<td>User Acceptance Test Plan</td>
<td>Business Project Manager</td>
</tr>
<tr>
<td>Construction</td>
<td></td>
<td></td>
<td></td>
<td>System Test Plan</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Ongoing</td>
<td>M</td>
<td></td>
<td></td>
<td>Issues List</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Ongoing</td>
<td></td>
<td></td>
<td></td>
<td>Risk List</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Ongoing</td>
<td>M</td>
<td></td>
<td></td>
<td>Project Plan / Iteration Plan</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Ongoing</td>
<td></td>
<td></td>
<td></td>
<td>Phase Assessment Review</td>
<td>Project Manager</td>
</tr>
<tr>
<td>Ongoing</td>
<td>M</td>
<td></td>
<td></td>
<td>Corporate Report Card</td>
<td>Business Project Manager, (A)</td>
</tr>
</tbody>
</table>

Key Roles/Owners of RUP Artifacts

<table>
<thead>
<tr>
<th>Key Role</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Sponsor</td>
<td>• Establishes the project funding and periodically review the spending progress against the Investment Opportunity expectations. They consistently champion the project and associated changes, as well as communicate project progress to Corporate leaders.</td>
</tr>
</tbody>
</table>
| Business Lead      | • Provides project leadership and overall business perspective. This role is responsible for managing the project risk and working with the team to ensure appropriate communication of risk mitigation.  
• Represents the team to stakeholders and management and influences the strategic and tactical business decisions pertaining to the project product. This role’s overall goal is to ensure the business expectations are achieved on time and on budget. |
| Business Project Manager | • Allocates resources, shapes priorities, coordinates interactions with customers and users, and generally keeps the project team focused on the right goal. The project manager also establishes a set of practices that ensure the integrity and quality of project artifacts. In addition, the Business Project Manager plans and conducts the formal review of the use-case model.  
• Leads and coordinates requirements elicitation and use-case modeling by outlining the system's functionality and delimiting the system; for example, establishing what actors and use cases exist and how they interact. In addition, this role details the specification of a part of the organization by describing the workflow of one or several business use cases. |
| Technology Project Manager | • Allocates resources, shapes priorities, coordinates interactions with customers and users, and generally keeps the project team focused on the right goal. The technology project manager also establishes a set of practices that ensure the integrity and quality of project artifacts. |
| Architect          | • Leads and coordinates technical activities and artifacts throughout the project.  
• The software architect establishes the overall structure for each architectural view: the decomposition of the view, the grouping of elements, and the interfaces between these major groupings. Therefore, in contrast to the other roles, the software architect's view is one of breadth as opposed to one of depth. |
Key Points

- RUP focuses on:
  - Iterative Controlled Development
  - Use Case Models for Business Requirements
  - Component Architecture
  - Risk Identification, Management & Mitigation

Part IX

*Introduction to Agile Software Engineering*
**Agile Software Engineering**

- **Agility**
  - “Ability to create and respond to change in order to profit in a turbulent business environment”
- **Agile Values**
  - Individual and interactions vs. processes and tools
  - Working software vs. comprehensive documentation
  - Customer collaboration vs. contract negotiation
  - Responding to change vs. following a plan

**Agile Software Development Ecosystems**

- Agile Software Development Ecosystems (ASDEs) vs. Traditional Software Development Methodologies
  - Chaordic perspective
    - Product goals are achievable but they are not predictable
    - Processes can aid consistency but they are not repeatable
  - Collaborative values and principles
  - Barely sufficient methodology
- Agilists are proponents of ASDEs
Part IX

Conclusion

Course Assignments

- Individual Assignments
  - Reports based on case studies / class presentations

- Project-Related Assignments
  - All assignments (other than the individual assessments) will correspond to milestones in the team project.
  - As the course progresses, students will be applying various methodologies to a project of their choice. The project and related software system should relate to a real-world scenario chosen by each team. The project will consist of inter-related deliverables which are due on a (bi-) weekly basis.
  - There will be only one submission per team per deliverable and all teams must demonstrate their projects to the course instructor.
  - A sample project description and additional details will be available under handouts on the course Web site.
Course Project

- Project Logistics
  - Teams will pick their own projects, within certain constraints: for instance, all projects should involve multiple distributed subsystems (e.g., web-based electronic services projects including client, application server, and database tiers). Students will need to come up to speed on whatever programming languages and/or software technologies they choose for their projects - which will not necessarily be covered in class.
  - Students will be required to form themselves into "pairs" of exactly two (2) members each; if there is an odd number of students in the class, then one (1) team of three (3) members will be permitted. There may not be any "pairs" of only one member! The instructor and TA(s) will then assist the pairs in forming "teams", ideally each consisting of two (2) "pairs", possibly three (3) pairs if necessary due to enrollment, but students are encouraged to form their own 2-pair teams in advance. If some students drop the course, any remaining pair or team members may be arbitrarily reassigned to other pairs/teams at the discretion of the instructor (but are strongly encouraged to reform pairs/teams on their own). Students will develop and test their project code together with the other member of their programming pair.

Readings

- Assignment #1
  - Team Project proposal (format TBD in class)
  - Presentation topic proposal (format TBD in class)
- Readings
  - Slides and Handouts posted on the course web site
  - Documentation provided with business and application modeling tools
  - SE Textbook: Chapters 1 & 2
- Project Frameworks Setup (ongoing)
  - As per references provided on the course Web site
Next Session:
Software Development Lifecycles (SDLCs)
Part I

- Lifecycle Phases
- Traditional Lifecycle Models
- Alternative Techniques
- Homework #1
- Project #1