Software Engineering

Session 1 – Main Theme
Software Engineering Fundamentals
Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

2

Who am I?

- Profile -

- 30 years of experience in the Information Technology Industry, including twelve years of experience working for leading IT consulting firms such as Computer Sciences Corporation
- PhD in Computer Science from University of Colorado at Boulder
- Past CEO and CTO
- Held senior management and technical leadership roles in many large IT Strategy and Modernization projects for fortune 500 corporations in the insurance, banking, investment banking, pharmaceutical, retail, and information management industries
- Contributed to several high-profile ARPA and NSF research projects
- Played an active role as a member of the OMG, ODMG, and X3H2 standards committees and as a Professor of Computer Science at Columbia initially and New York University since 1997
- Proven record of delivering business solutions on time and on budget
- Original designer and developer of jcrew.com and the suite of products now known as IBM InfoSphere DataStage
- Creator of the Enterprise Architecture Management Framework (EAMF) and main contributor to the creation of various maturity assessment methodology
- Developed partnerships between several companies and New York University to incubate new methodologies (e.g., EA maturity assessment methodology developed in Fall 2008), develop proof of concept software, recruit skilled graduates, and increase the companies' visibility

3

How to reach me?

	Cell	(212) 203-5004
	Email	jcf@cs.nyu.edu
	AIM, Y! IM, ICQ	jcf2_2003
	MSN IM	jcf2_2003@yahoo.com
LinkedIn	LinkedIn	http://www.linkedin.com/in/jcfranchitti
twitter	Twitter	http://twitter.com/jcfranchitti
skype	Skype	jcf2_2003@yahoo.com

4

What is the class about?

- Course description and syllabus:
  - » http://www.nyu.edu/classes/jcf/g22.2440-001/
  - » http://www.cs.nyu.edu/courses/spring13/G22.2440-001/
- Textbooks:
  - » ***Software Engineering: A Practitioner's Approach***

Roger S. Pressman
McGraw-Hill Higher International
ISBN-10: 0-0712-6782-4, ISBN-13: 978-00711267823, 7th Edition (04/09)

  - » http://highered.mcgraw-hill.com/sites/0073375977/information_center_view0/
  - » http://highered.mcgraw-hill.com/sites/0073375977/information_center_view0/table_of_contents.html

5

Icons / Metaphors

Information

Common Realization

Knowledge/Competency Pattern

Governance

Alignment

Solution Approach

6

Helpful Preliminary Knowledge

- Business Process Modeling (BPM)
- 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, C#)
- Note: Knowledge of BPMN, UML or a specific programming language is not required

7

Course Objectives (1/3)

- 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

8

Course Objectives (2/3)

- Expand students' familiarity with mainstream languages used to model and analyze processes and object designs (e.g., BPMN, 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.

9

Course Objectives (3/3)

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

10

Software Requirements

- Microsoft Windows XP (Professional Ed.) / Vista / 7
- 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

11

Agenda

1. Instructor and Course Introduction
2. Software Engineering Fundamentals
3. Towards a Pattern-Driven SE Methodology
4. Summary and Conclusion

12

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

13

What is Software? (1/2)

Software is:

(1)instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information;

(3) documentation that describes the operation and use of the programs.

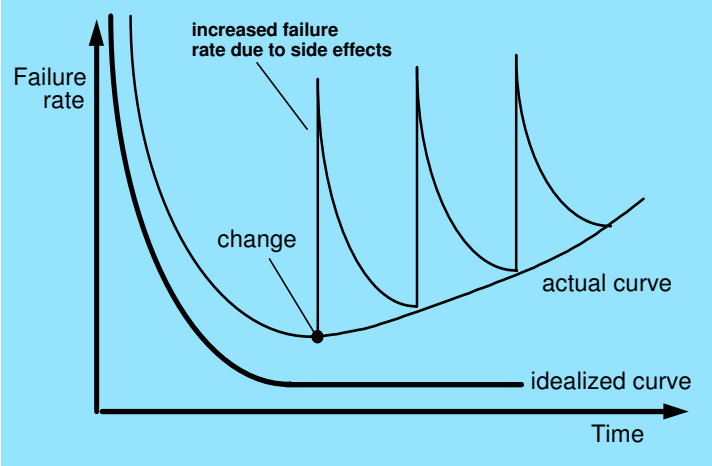
14

What is Software? (2/2)

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

15

Wear vs. Deterioration



16

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 **G**ross **N**ational **P**roduct. 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.

17

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

18

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

19

Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

20

Software—New Categories

- Open world computing - pervasive, distributed computing
- Ubiquitous computing - wireless networks
- Netsourcing - the Web as a computing engine
- Open source - ”free” source code open to the computing community (a blessing, but also a potential curse!)
- Also …
  - » Data mining
  - » Grid computing
  - » Cognitive machines
  - » Software for nanotechnologies

21

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology
- software must be enhanced to implement new business requirements
- software must be extended to make it interoperable with other more modern systems or databases
- software must be re-architected to make it viable within a network environment

22

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

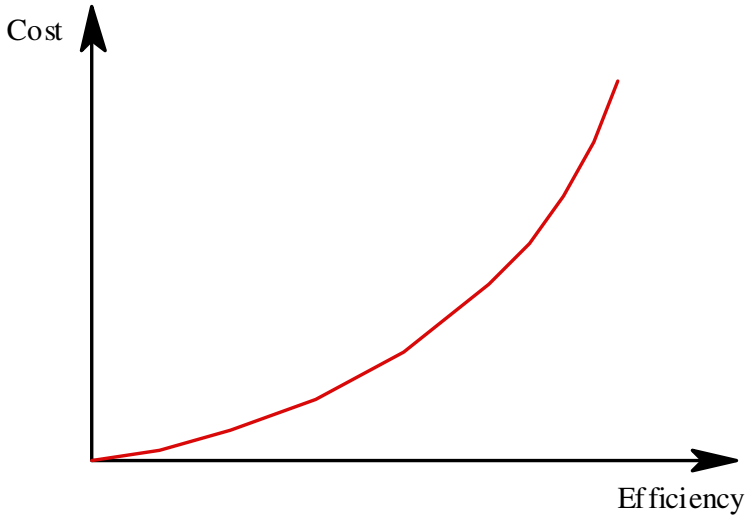
23

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

24

Efficiency Costs



25

Characteristics of WebApps (1/2)

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a "24/7/365" basis.
- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.

26

Characteristics of WebApps (2/2)

- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application
- **Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel

27

Summary of Sub-Section's Key Points

- 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 include maintainability, dependability, efficiency and usability

28

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

29

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
- Software process must be explicitly modeled if it is to be managed

30

Process Characteristics (1/2)

- 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?

31

Process Characteristics (2/2)

- 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?

32

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

33

Software Process Models Characteristics

- 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

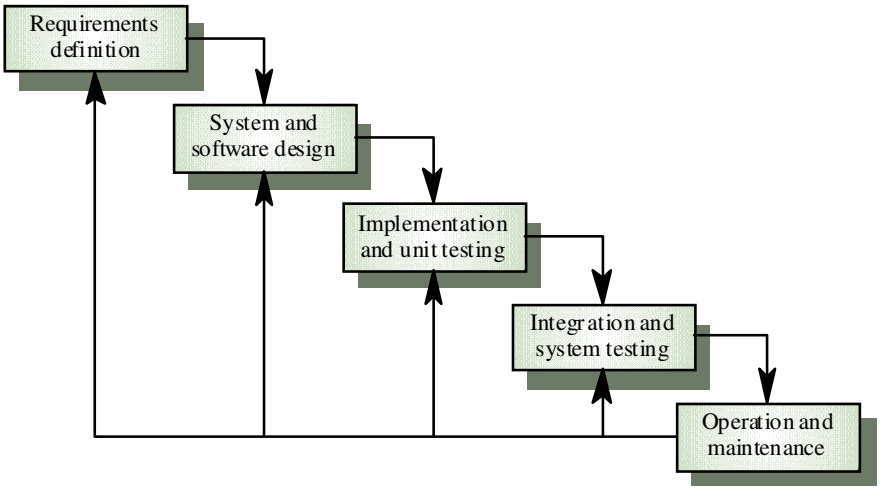
34

Generic Software Process Models

- 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

35

Waterfall Model



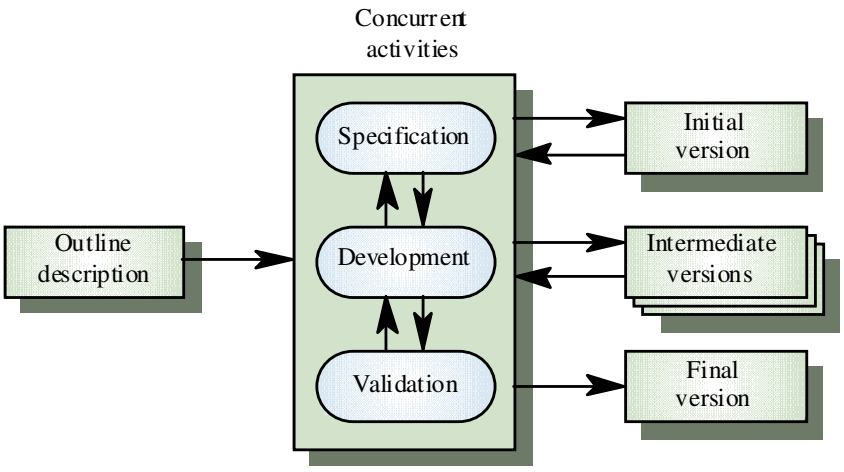
36

Waterfall Model Characteristics and Limitations

- 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

37

Evolutionary Development



38

Evolutionary Development Characteristics

- 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

39

Evolutionary Development Limitations

- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Requires 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

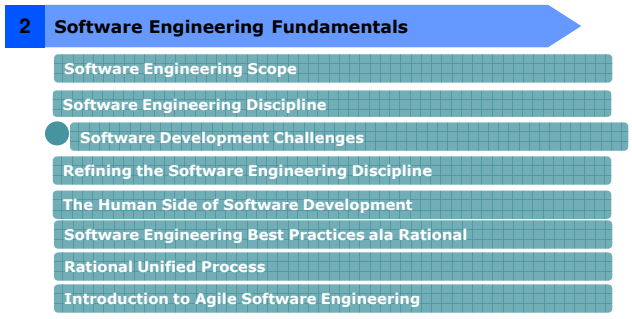
40

Summary of Sub-Section's 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

41

Agenda – Software Engineering Fundamentals



42

Inherent Risks (http://www.ibm.com/developerworks/rational/library/1719.html)

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

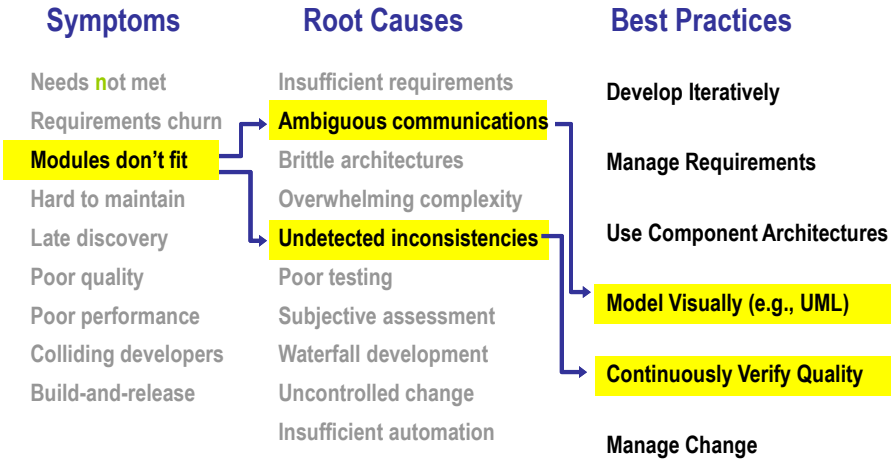
43

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

44

Trace Symptoms to Root Causes



45

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

46

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

47

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

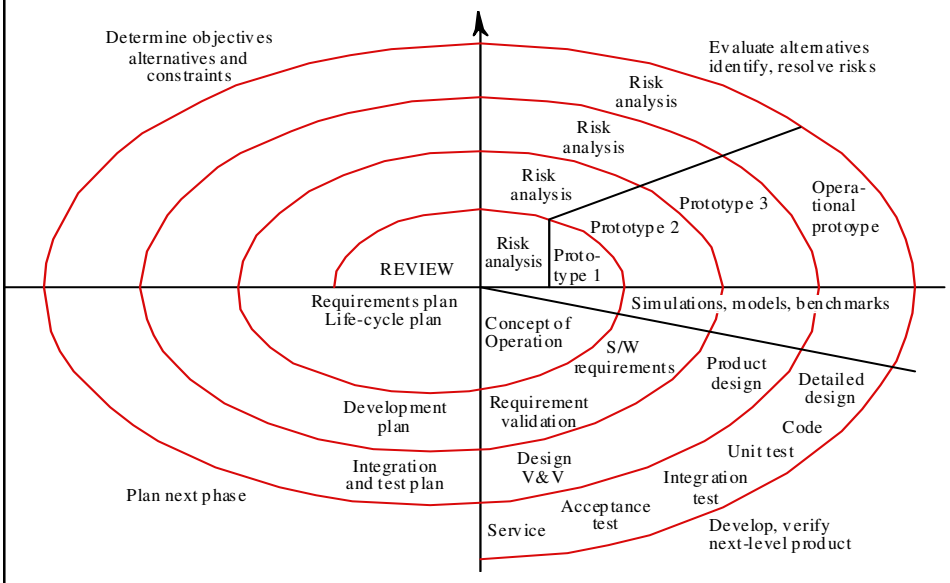
48

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 should be used for high-risk specifications
- Waterfall model should be used for well-understood developments

49

Spiral Model of the Software Process



50

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

51

Template for a Spiral Round

- Quality Improvement Focus
  - Objectives
  - Constraints
  - Alternatives
- Risk Reduction Focus
  - Risk Assessment
  - Risk resolution
- Plan-Do-Check-Act (PDCA) Approach
  - Results
  - Plans
  - Commitment

52

Quality Improvement Focus

- 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

53

Risk Reduction Focus

- Risk Assessment
  - 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

54

PDCA 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

55

Template for a Spiral Round at Work - Catalogue Spiral (1/3)

- Quality Improvement Focus
  - 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

56

Template for a Spiral Round at Work - Catalogue Spiral (2/3)

- Risks Reduction Focus
  - Risks assessment
    - 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

57

Template for a Spiral Round at Work - Catalogue Spiral (3/3)

- PDCA 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

58

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

59

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

60

Spiral Model Limitations

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

61

Process Visibility as a Process Model Metric

- 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 places constraints on process iterations
  - The time taken to review and approve documents is significant
- Waterfall model is still the most widely used deliverable-based model

62

Sample Set of Waterfall Model Documents

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

63

Process Model Visibility

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

64

Summary of Sub-Section's Key Points

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

65

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

66

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

67

Ethical Issues

- Confidentiality
- Competence
- Intellectual property rights
- Computer misuse

68

Summary of Sub-Section's Key Points

- Software engineers have ethical, social and professional responsibilities

69

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

70

Section Outline

- Identify Steps for Understanding and Solving Software Engineering Problems
- Explain the IBM Rational “Six Best Practices”

71

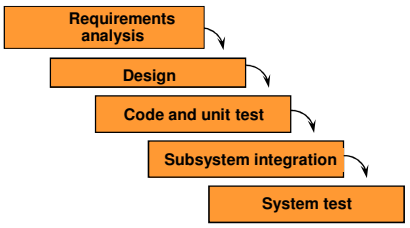
Practice 1: Develop Iteratively

Best Practices
Process Made Practical

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

72

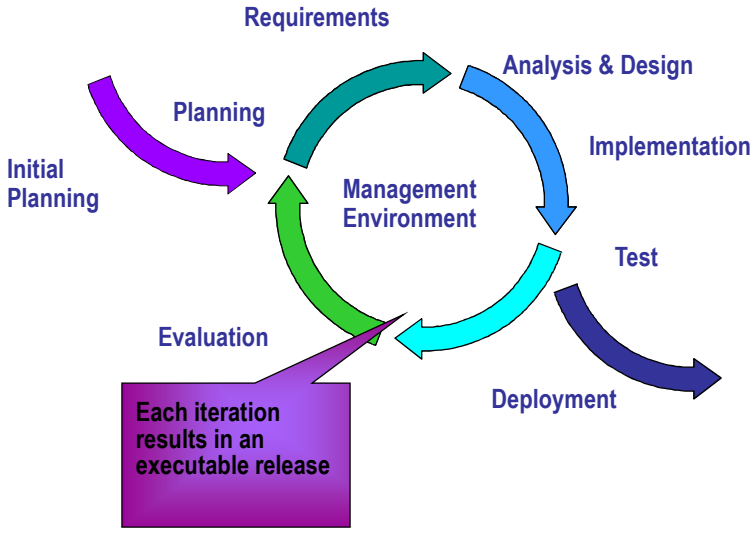
Waterfall Development Characteristics



- 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

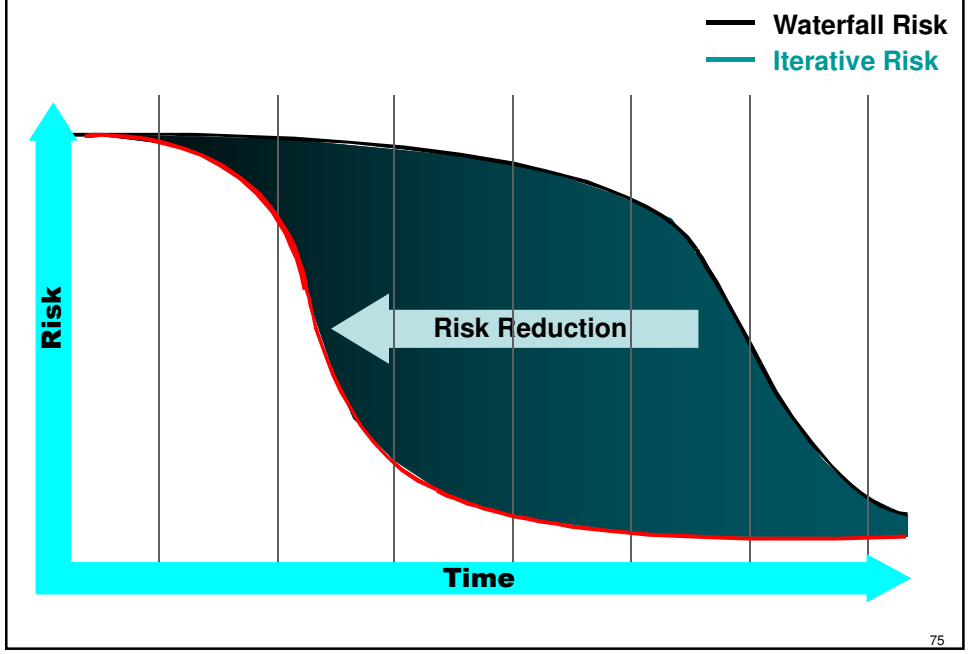
73

Iterative Development Produces Executable Releases



74

Risk Profiles



75

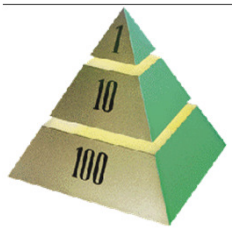
Practice 2: Manage Requirements

Best Practices
Process Made Practical

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

76

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.

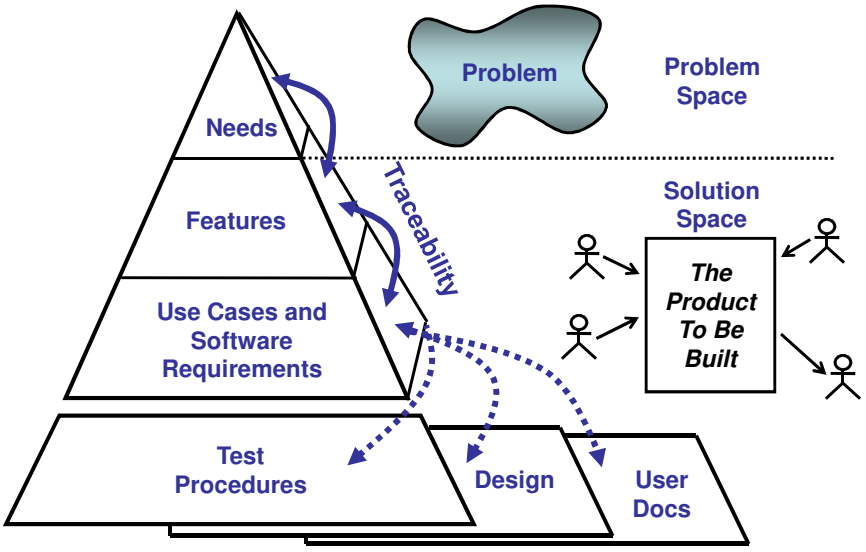
77

Aspects of Requirements Management

- Analyze the Problem
- Understand User Needs
- Define the System
- Manage Scope
- Refine the System Definition
- Build the Right System

78

Map of the Territory



79

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

80

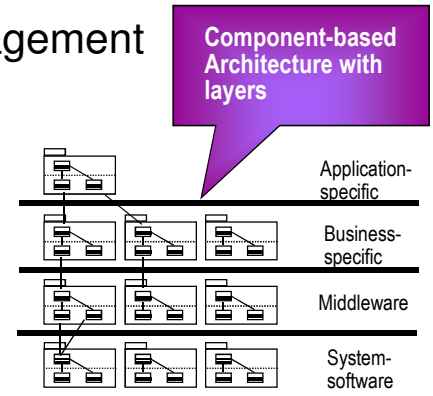
Resilient, Component-Based Architectures

- 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

81

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



82

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

83

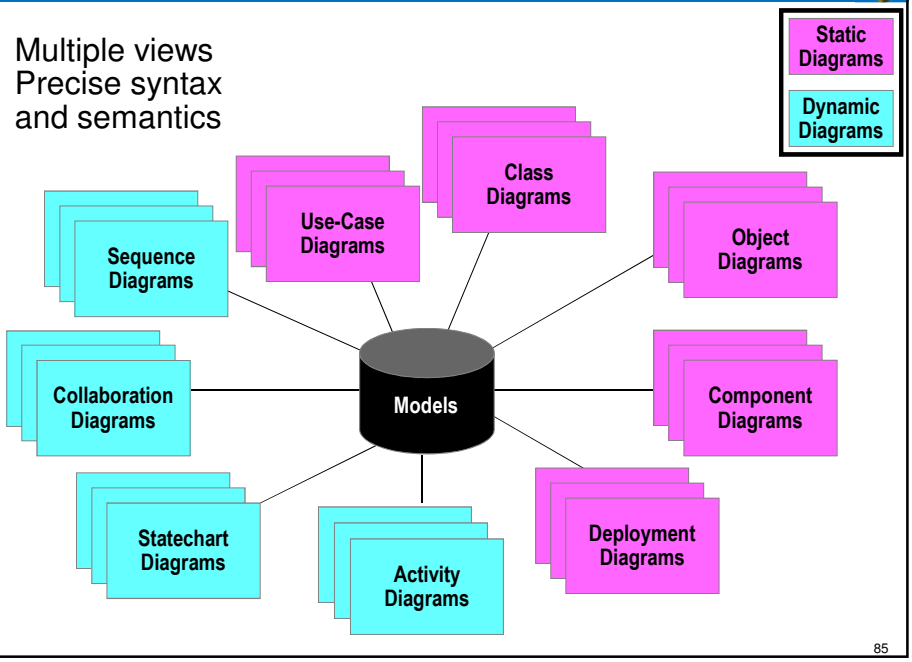
Why Model Visually?

- 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

84

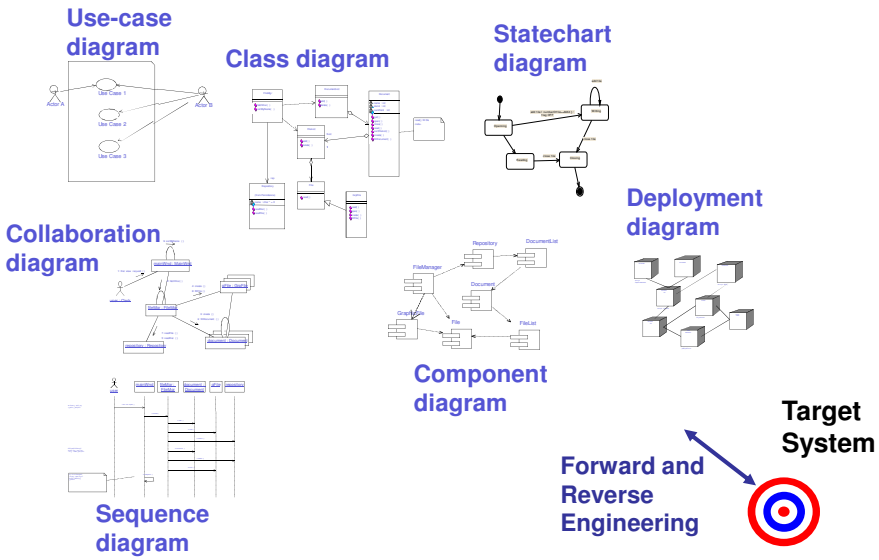
Visual Modeling with UML 1.X

- Multiple views
- Precise syntax and semantics



85

Visual Modeling Using UML 1.X Diagrams



86

UML 1.X Notation Baseline

Diagram Name	Type	Phase
Use Case	Static*	Analysis
Class	Static	Analysis
Activity	Dynamic**	Analysis
State-Transition	Dynamic	Analysis
Event Trace (Interaction)	Dynamic	Design
Sequence	Dynamic	Design
Collaboration	Dynamic	Design
Package	Static	Delivery
Deployment	Dynamic	Delivery

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

87

UML 1.X Diagrams

UML 1.X 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 to organize and manage your application modules
  - Packages
  - Subsystems
  - Models

Source: http://www.omg.org/gettingstarted/what_is_uml.htm

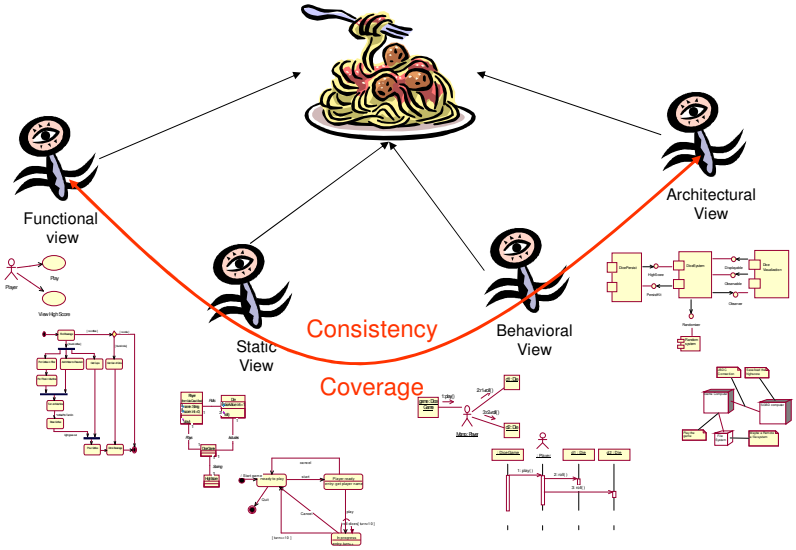
88

UML 1.X Views

- Approach
  - UML 1.X 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

89

Need to Maintain Consistency and Coverage Across UML 1.X Views



90

New in UML 2.X (1/2) (http://www.omg.org/gettingstarted/what_is_uml.htm)

- UML 2.X Profiles
  - The new language goes well beyond the Classes and Objects well-modeled by UML 1.X to add the capability to represent not only behavioral models, but also architectural models, business process and rules, and other models used in many different parts of computing and even non-computing disciplines
- Nested Classifiers
  - Every model building block (e.g., classes, objects, components, behaviors such as activities and state machines) is a *classifier*
    - A set of classes may be nested inside the component that manages them, or a behavior (such as a state machine) may be embedded inside the class or component that implements it
    - Capability may be used to build up complex behaviors from simpler ones (i.e., the capability that defines the Interaction Overview Diagram)
    - Can layer different levels of abstraction in multiple ways:
      - For example, you can build a model of your Enterprise, and zoom in to embedded site views, and then to departmental views within the site, and then to applications within a department
      - Alternatively, you can nest computational models within a business process model. OMG's Business Enterprise Integration Domain Task Force (BEI DTF) is currently working on several interesting new standards in business process and business rules

91

New in UML 2.X (2/2) (http://www.omg.org/gettingstarted/what_is_uml.htm)

- Improved Behavioral Modeling
  - In UML 1.X, the different behavioral models were independent, but in UML 2.0, they all derive from a fundamental definition of a behavior (except for the Use Case, which is subtly different but still participates in the new organization)
- Improved relationship between Structural and Behavioral Models
  - UML 2.0 makes it possible to designate that a behavior represented by (for example) a State Machine or Sequence Diagram is the behavior of a class or a component
- Object Constraint Language (OCL) and Action Semantics
  - During the upgrade process, several additions to the language were incorporated into it, including the Object Constraint Language (OCL) and Action Semantics.

92

Practice 5: Continuously Verify Quality

Best Practices
Process Made Practical

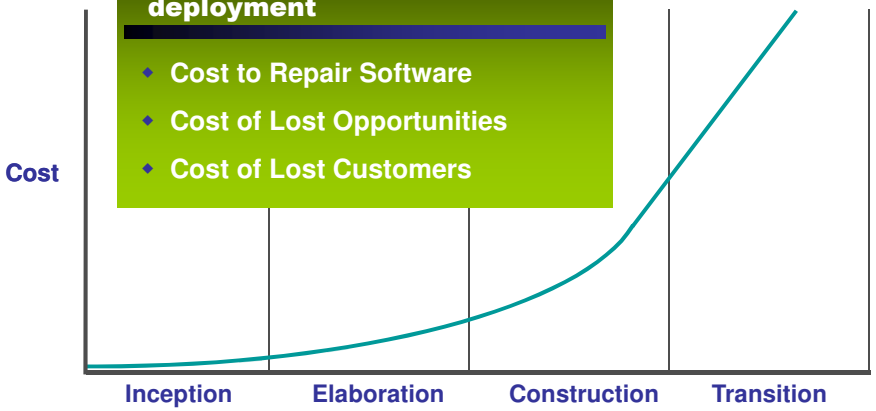
Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

93

Continuously Verify Software Quality

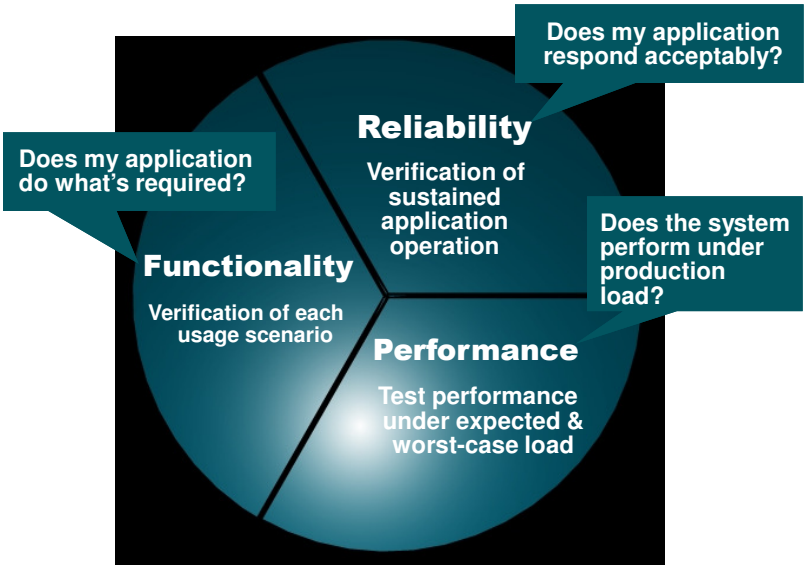
Software problems are 100 to 1000 times more costly to find and repair after deployment

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



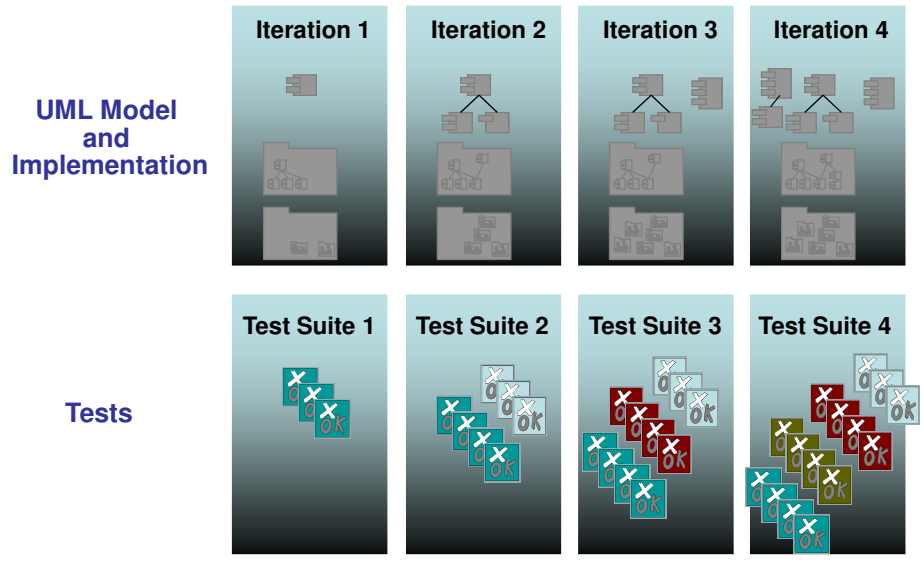
94

Test All Dimensions of Software Quality



95

Test Each Iteration



96

Practice 6: Manage Change

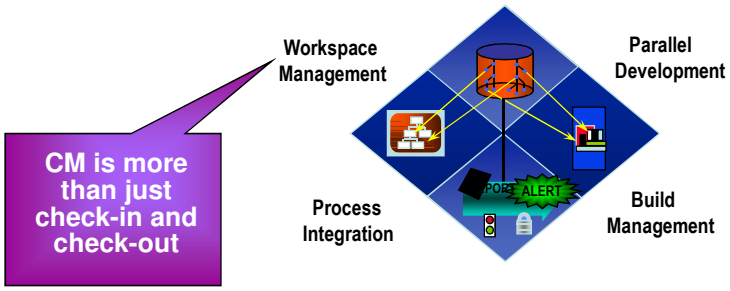
Best Practices
Process Made Practical

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

97

What Do You Want to Control?

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



98

Aspects of a Configuration Management (CM) System

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

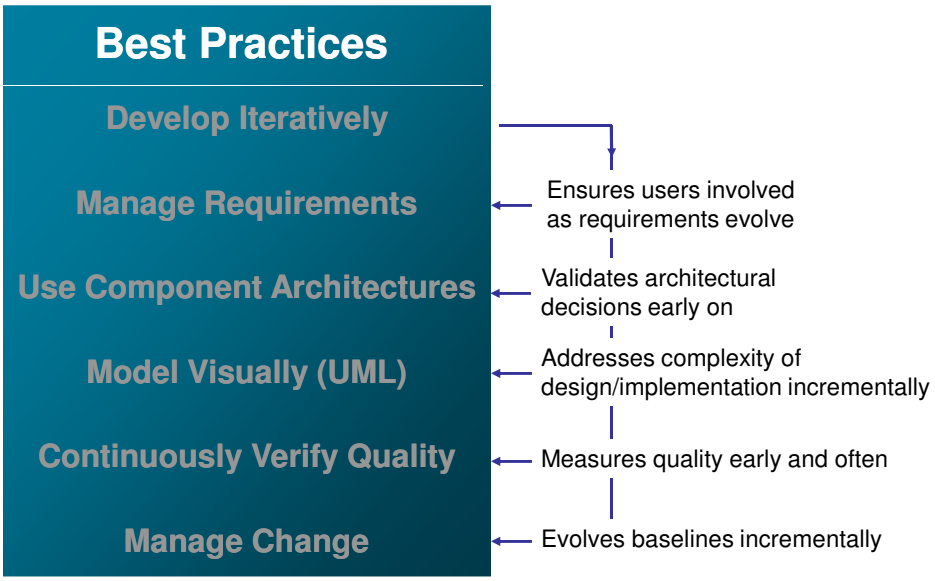
99

Unified Change Management

- Management across the lifecycle
  - System
  - Project Management
- Activity-Based Management
  - Tasks
  - Defects
  - Enhancements
- Progress Tracking
  - Charts
  - Reports

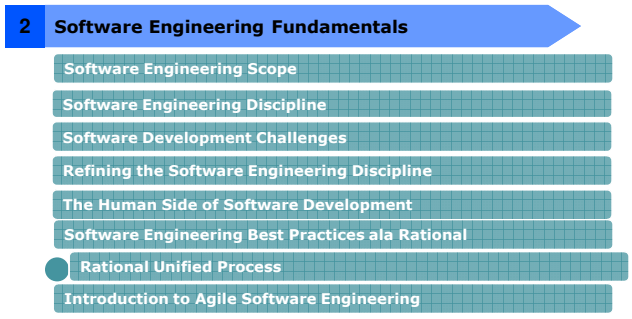
100

Best Practices Reinforce Each Other



101

Agenda – Software Engineering Fundamentals



102

Section Outline

- Present the IBM Rational Unified Process within the context of the Six Best Practices covered in the previous sub-section

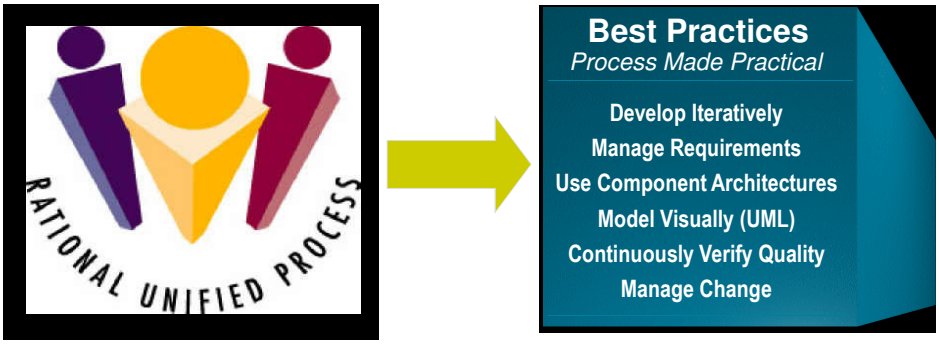
103

Foundations of RUP

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

104

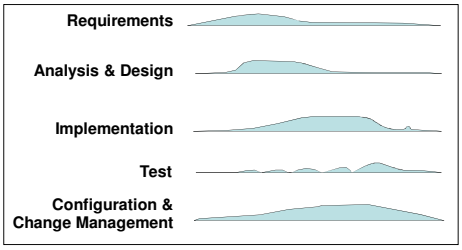
RUP Best Practices Implementation



105

Achieving Best Practices

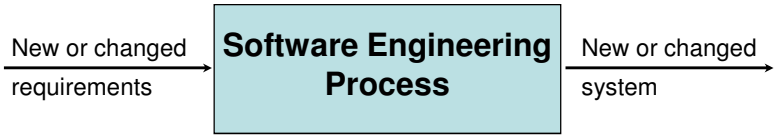
- Iterative Approach
- Guidance for activities and work products (artifacts)
- Process focus on architecture
- Use cases which drive design and implementation
- Models which abstract the system



106

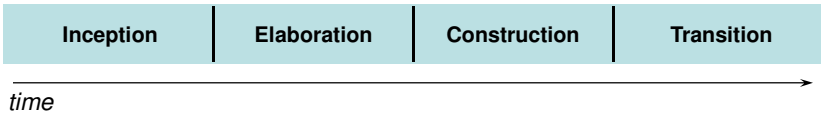
A Team-Based Definition of Process

- A process defines **Who** is doing **What**, **When** and **How** to reach a certain goal.



107

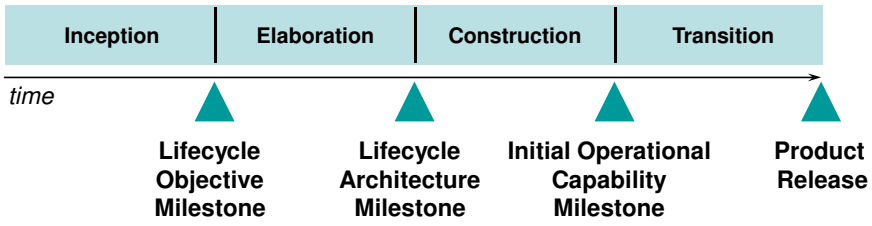
Process Structure - Lifecycle Phases



- 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

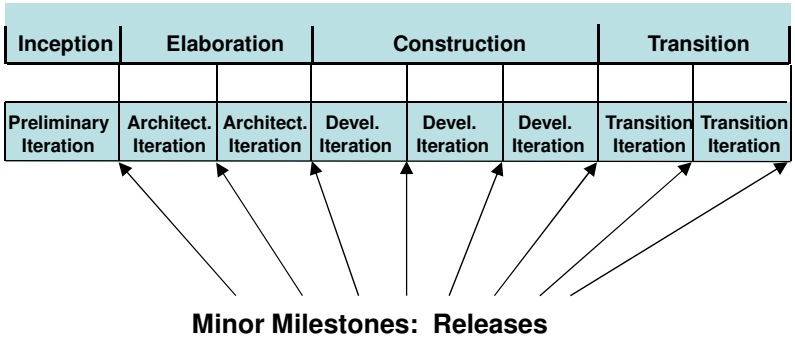
108

Phase Boundaries Mark Major Milestones



109

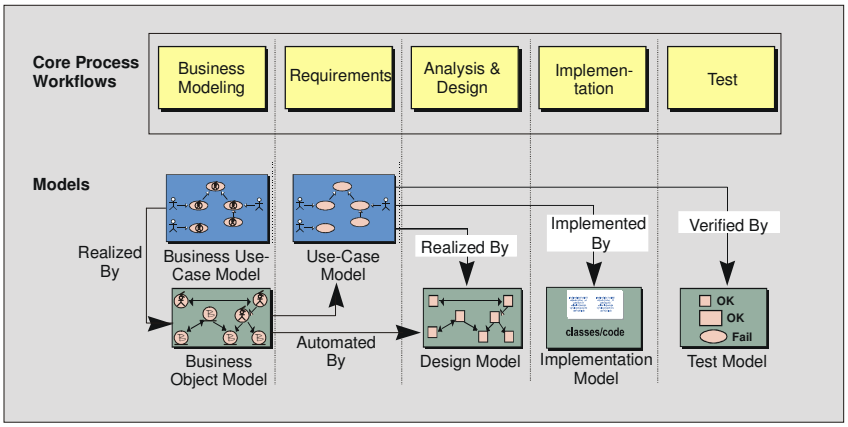
Iterations and Phases



An **iteration** is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external)

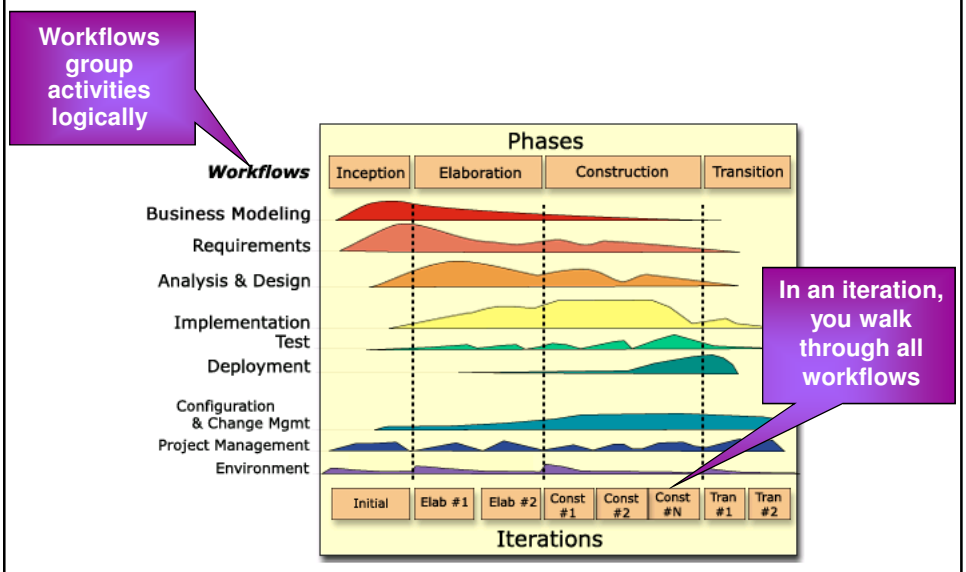
110

Workflows Produce Models



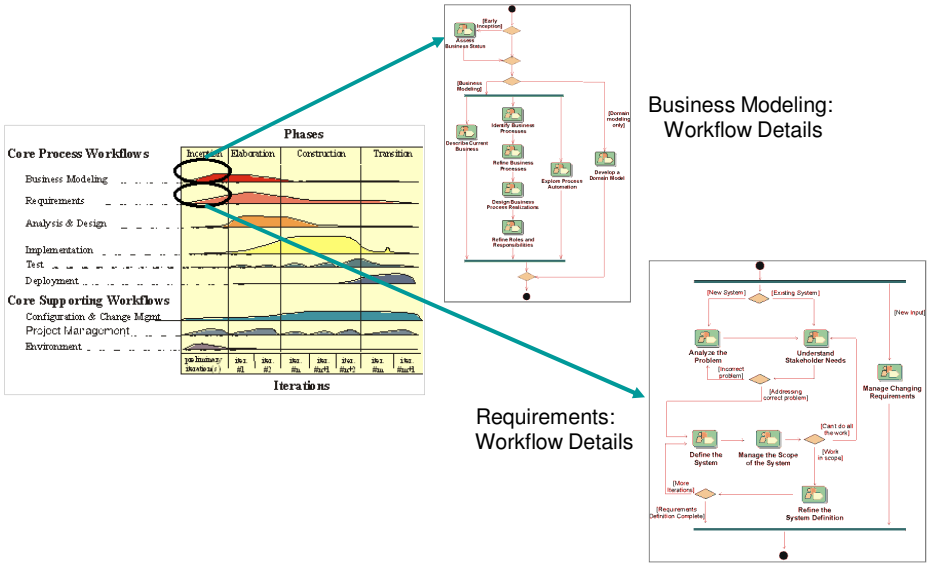
111

Bringing It All Together: The Iterative Approach



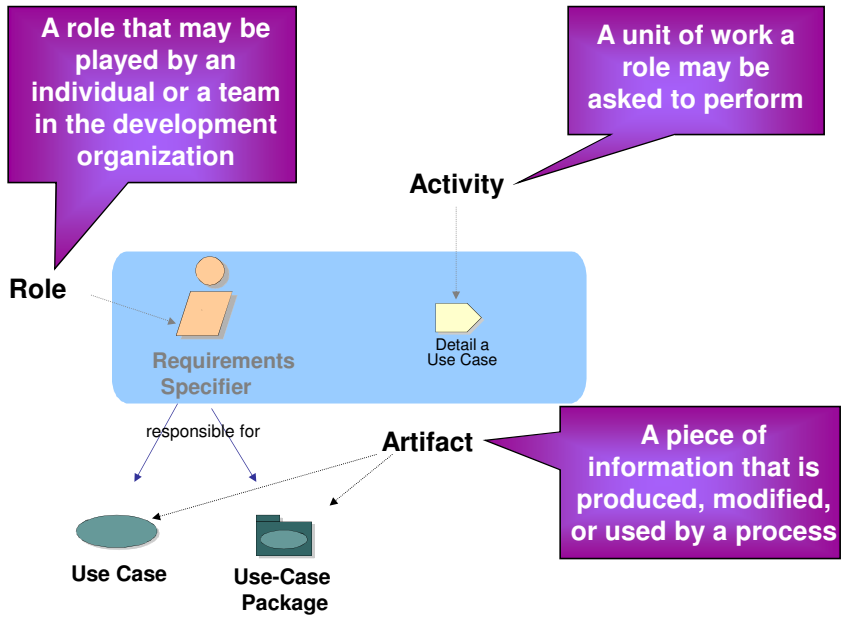
112

Workflows Guide Iterative Development



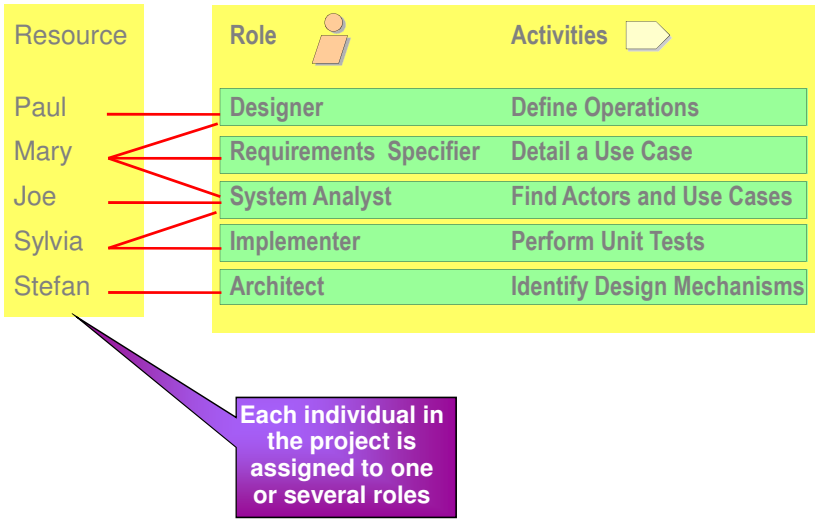
113

Notation



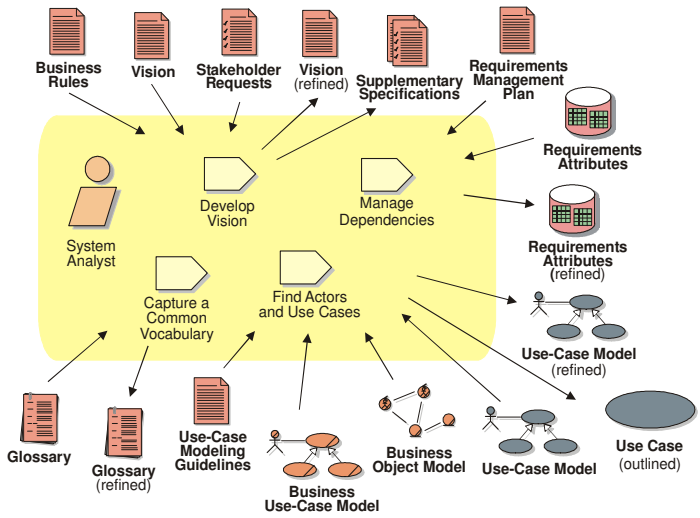
114

Roles Are Used for Resource Planning



115

Roles Perform Activities and Produce Artifacts



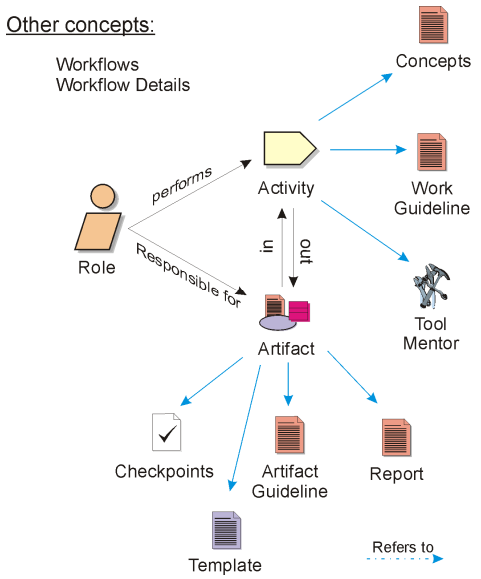
Example Requirements:

Workflow Detail

"Define the System"

116

Overview of Rational Unified Process Concepts



117

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

118

Sample RUP Artifacts Definition

Artifacts	Definitions
Investment Concept Statement Business Case	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
Vision	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
Stakeholders Requests	Captures all requests made on the project from stakeholders
Technology Governance Questionnaire	Assesses the impact of all development projects introducing significant architectural or high-level design changes
Use Case Specifications	Defines the functional requirements for the system with use case diagrams
Supplementary Specifications	Defines the nonfunctional requirements of the system
Software Architecture Document	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)
User Acceptance Test Plan	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
System Test Plan	Outlines and communicates the objectives of the testing effort to gain acceptance and approval from the stakeholders
Corporate Report Card	Provides measurement and explanation of variances between actual and expected project performance and informs management of project issues (High Risk, High Impact)
Issues List	Entails the documentation, review, resolution, and follow-up of project issues
Risk List	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
Project Plan / Iteration Plan	Details the specific tasks that must be completed by each team member in order to complete a project
Phase Assessment Review	Establishes criteria for determining whether or not a project is ready to move from one phase to the next phase

119

Sample RUP Core Artifacts

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

A = Approver

120

Sample Key Roles/Owners of RUP Artifacts

Key Role	Definition
Business Sponsor	▪ 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.
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.

121

Summary of Sub-Section's Key Points

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

122

Agenda – Software Engineering Fundamentals

2 **Software Engineering Fundamentals**

- 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
- Introduction to Agile Software Engineering

123

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

124

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

125

Agenda

1. Instructor and Course Introduction
2. Software Engineering Fundamentals
3. Towards a Pattern-Driven SE Methodology
4. Summary and Conclusion

126

Section Objectives

- Describe the limitations of legacy and best practice SDLC methodologies
- Suggest the improved approach that is covered in the course
- Discuss the approach to follow for the class project

127

Limitations of Legacy SE Methodologies

- Focused on software solutions development
- Driven by processes
  - Not driven by architecture and/or best practices altogether (other than initially)
- Focus is on scope, time, cost, and quality
  - customer input sparsely considered
- Metaphor:
  - “an algorithm without a centralized data structure to operate on”

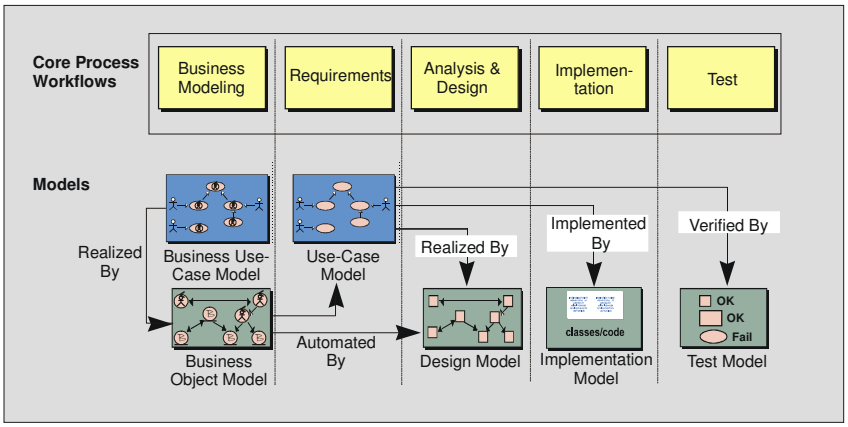
128

Limitations of RUP Approach

- Focused on software solutions development
- Driven by best practices
  - Driven by workflows (and tools)
- Focus is on scope, time, and cost
  - Customer assesses quality and drive change
  - Deliver quality software on-time & on-budget
    - By enforcing a best practice process that manages change
    - By following a PDCA approach were individuals play various roles in the overall process
- Gap between Architecture-Driven approach and Use-Case Driven Modeling
  - A "top-down" architectural approach

129

Illustrating the RUP "Gap"



- Going from business requirements to use cases requires non-trivial input that is hard/impossible to predict

130

Limitations of ASDE Approaches

- Focused on software solutions development
- Driven by best practices
  - Driven by collaboration between individuals
    - Interactions: customer/project team & intra-project team
  - Driven by change
- Focus is on quality (test-driven), time, and cost
  - Customer drives the scope
  - Deliver optimal quality software on-time & on-budget
    - By limiting the scope to facilitate change
    - By follow an MOB approach were individuals assume full leadership
- Architectural re-factoring becomes a nightmare
  - A “bottom-up architectural approach”

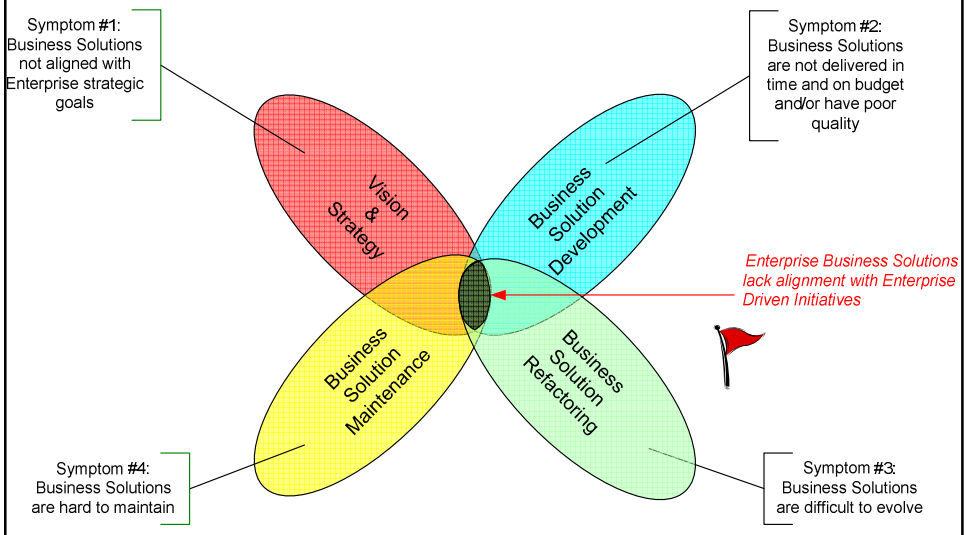
131

Agile Pattern-Driven Architecture (PDA) Approach

- Focused on business solutions development
  - » SDLC stands for “(Business) Solution Development LifeCycle”
- Seed the Architecture-Driven approach so it does not operate top-down or bottom-up
  - » Integrate the Architecture-Driven approach into standard and business specific architecture-driven workflows
    - e.g., AKDAR, GDM, SBAM, PEM, LSS (BPM pattern), CBM (SOA pattern)
  - » Use an agile workflow-driven approach rather than rigid processes
  - » Use architecture-driven approach from business strategy all the way down to product maintenance
  - » Subject individuals to ongoing transformation processes
- Flexible RUP-like or ASDE-like focus and introduces problem pattern set as an additional variable
- Need to deal with individuals reaction to the constant need to adapt to change
  - » Build conducive environments (e.g., game-metaphor, etc.)

132

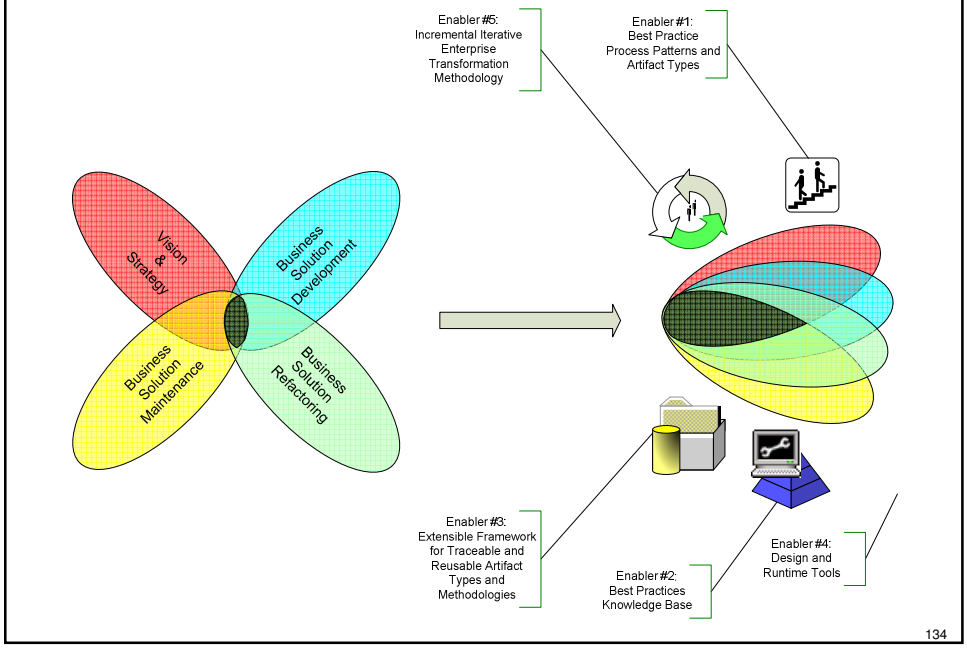
Enterprise Strategy and Business Solutions Alignment Problem



133

PDA Solution: Enterprise Architecture Management

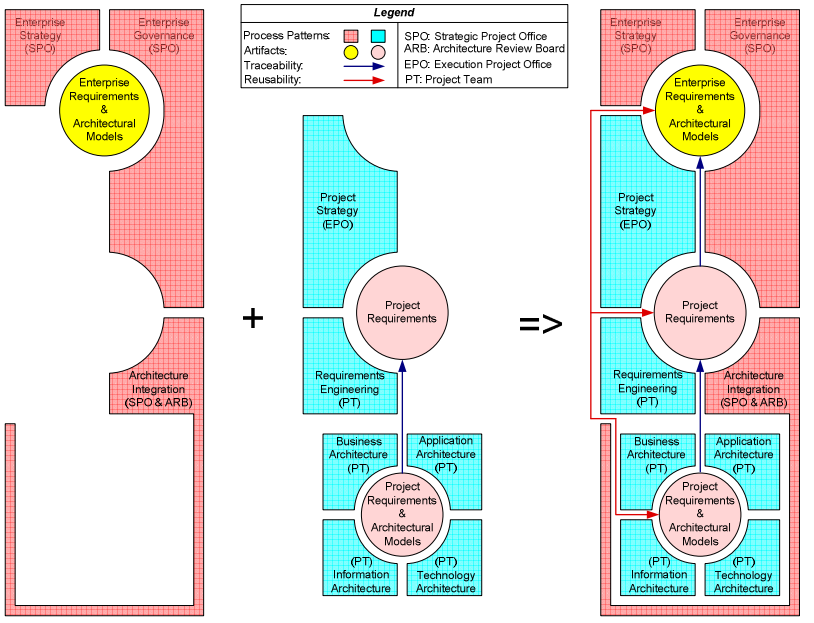
"Focusing on Business Model Improvements while Maintaining Enterprise Alignment"



134

Strategy Enablement Process Patterns and Artifact Types

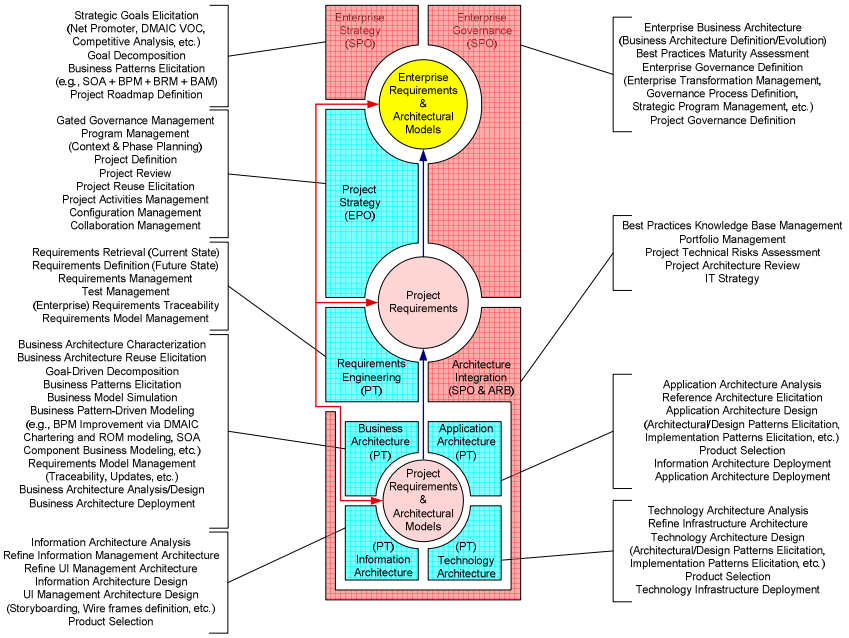
"Enabler #1"



135

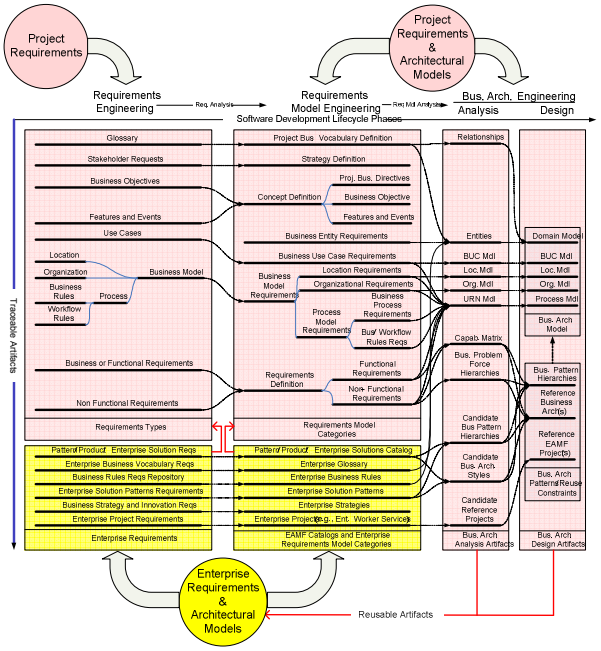
Strategy Enablement Process Patterns Detailed

A Process Pattern Leads to a Methodology Once Specific Activities are Chosen to Implement a Vision



136

Strategy Enablement Artifact Types Detailed



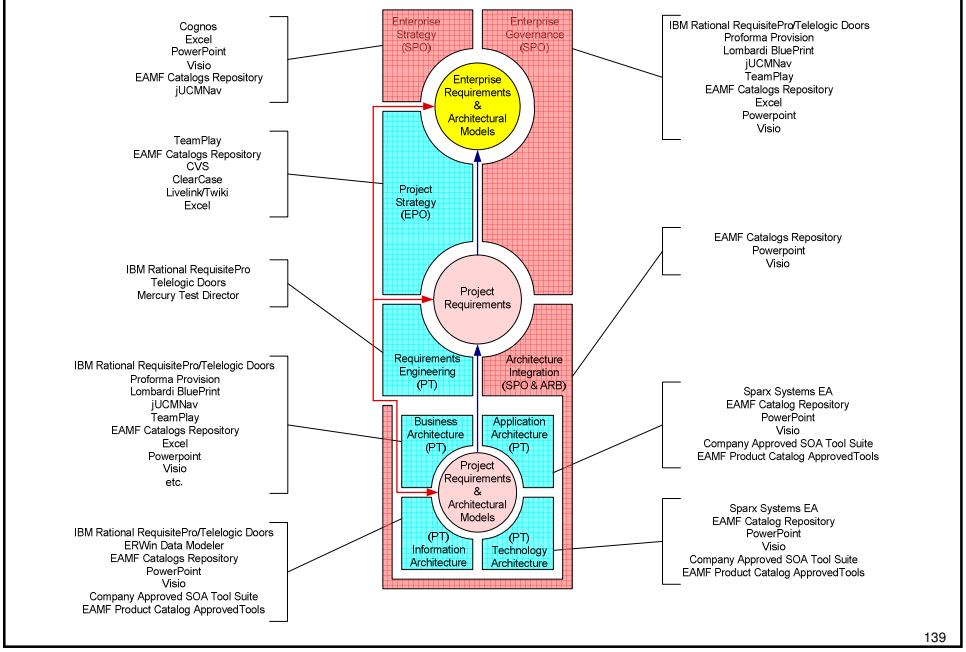
Extensible Framework and Best Practices Knowledge Base

"Enablers #2 and #3 (Sample)" – EAMF Framework Summary of Capabilities

- Extension of the TOGAF Industry Standard
  - http://www.opengroup.org/togaf/
- Differentiators:
  - Business Pattern Oriented Architecture (POA) orientation
    - Extensible methodology based on business solution patterns
    - Extensible knowledge foundation based on best practices and ongoing strategies and business solution development
  - Artifact Traceability Focus
  - Agile Activity-Driven Approach
    - Solution Development Lifecycle agnostic
  - Solution-Driven Approach
  - Tool Agnostic Approach

Strategy Enablement From a Tools Perspective

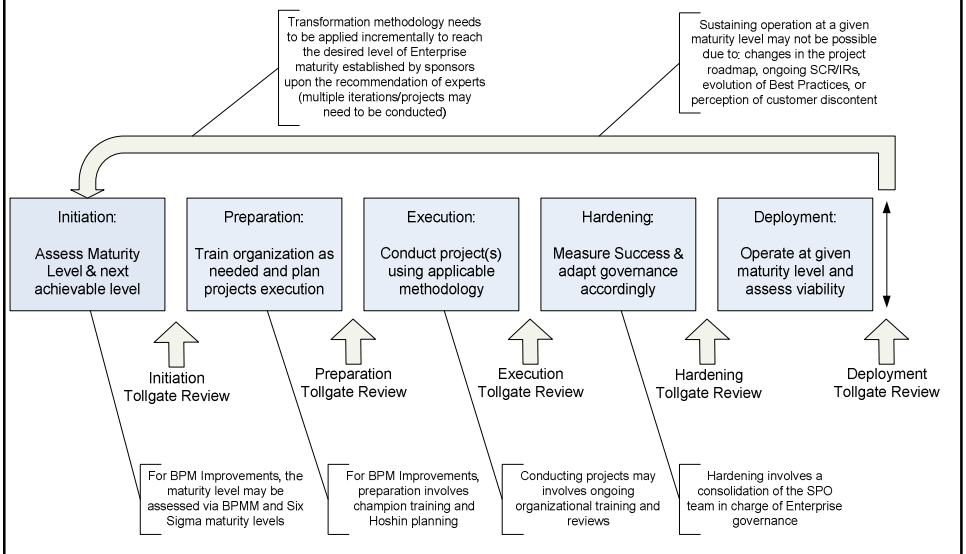
Enabler #4 (Sample): EAMF Framework Implementation



139

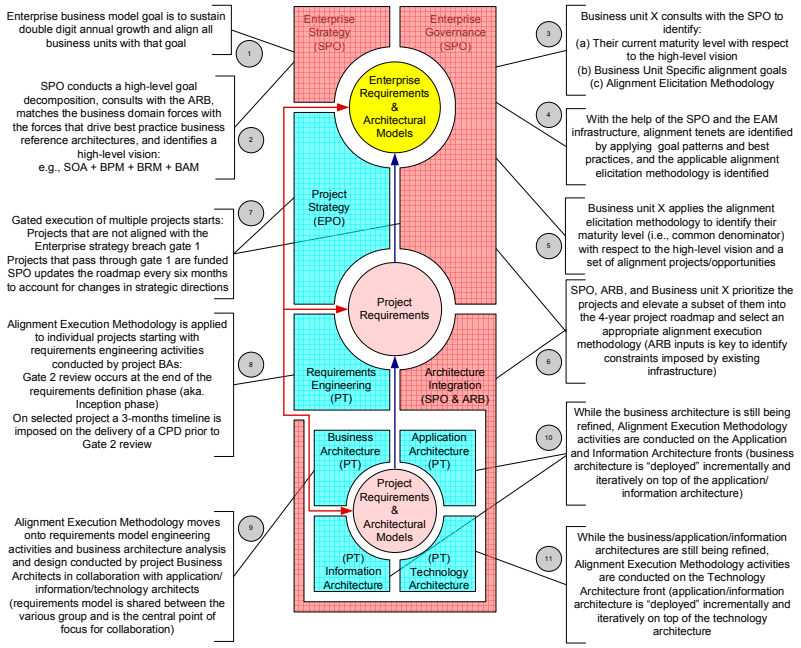
Incremental Iterative Enterprise Transformation Methodology

"Enabler #5"



140

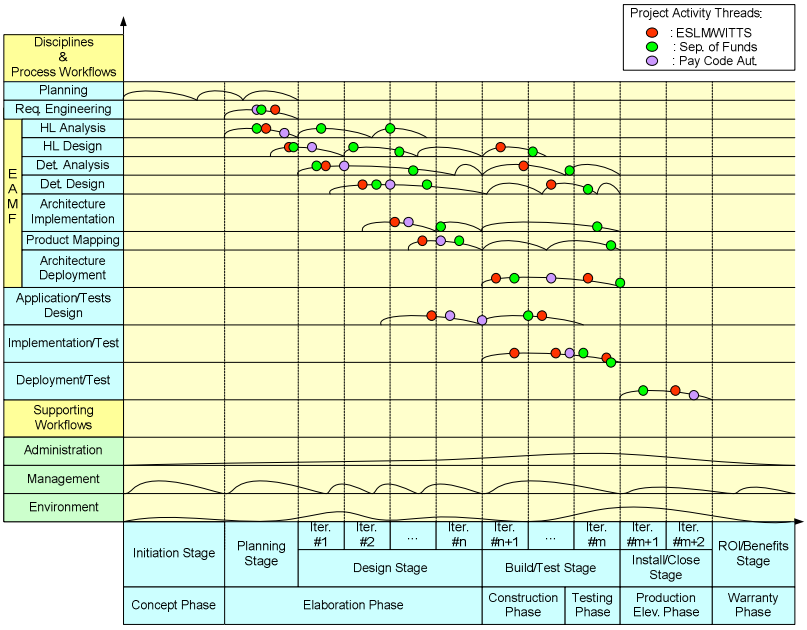
Strategy Enablement At Work



141

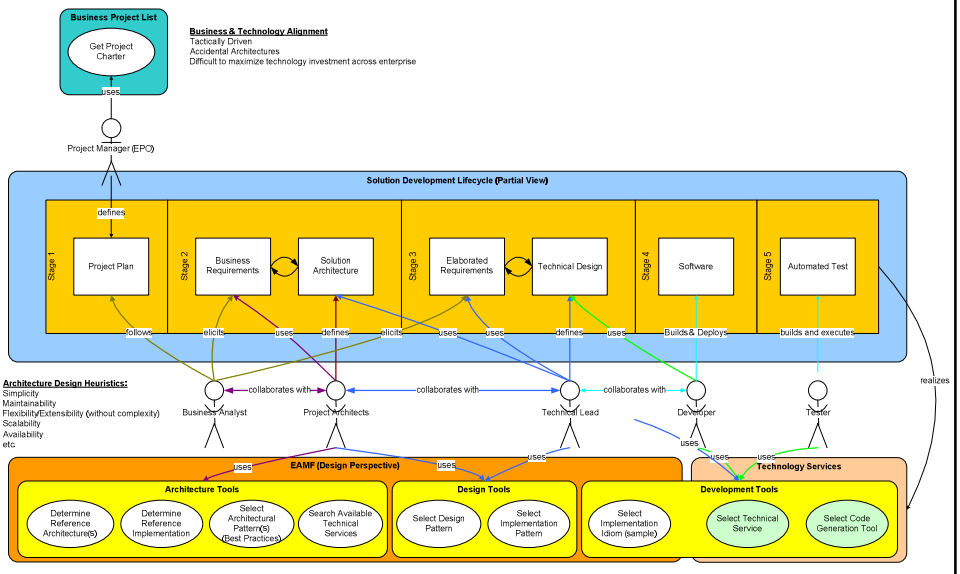
Enterprise Architecture Management

EAMF Activities Integrate Seamlessly with the Company X Project Lifecycle



142

Enterprise Architecture Management
Integration with the Company X Project Lifecycle all



143

Agenda

1. Instructor and Course Introduction
2. Software Engineering Fundamentals
3. Towards a Pattern-Driven SE Methodology
4. Summary and Conclusion

144

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

145

Team 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.

146

Team Project Approach - Overall

- Document Transformation methodology driven approach
  - » Strategy Alignment Elicitation
    - Equivalent to strategic planning
      - i.e., planning at the level of a project set
  - » Strategy Alignment Execution
    - Equivalent to project planning + SDLC
      - i.e., planning a the level of individual projects + project implementation
- Build a methodology Wiki & partially implement the enablers
- Apply transformation methodology approach to a sample problem domain for which a business solution must be found
- Final product is a wiki/report that focuses on
  - » Methodology / methodology implementation / sample business-driven problem solution

147

Team Project Approach – Initial Step

- Document sample problem domain and business-driven problem of interest
  - » Problem description
  - » High-level specification details
  - » High-level implementation details
  - » Proposed high-level timeline

148

Assignments & Readings

- Readings
  - » Slides and Handouts posted on the course web site
  - » Textbook: Chapter 1 & Part One-Chapter 2
- Assignment #1
  - » Team Project proposal (format TBD in class)
  - » Presentation topic proposal (format TBD in class)
- Project Frameworks Setup (ongoing)
  - » As per reference provided on the course Web site

149

Next Session: Software Development Lifecycles (SDLCs)

- Software Engineering Detailed
- Process Models
- Agile Development
- Software Engineering Knowledge
- Roles and Types of Standards
  - ISO 12207: Life Cycle Standard
  - IEEE Standards for Software Engineering Processes and Specifications
- Summary and Conclusion
  - Readings
  - Assignment #1
  - Course Project

150

Any Questions?

151