Introducing the Unified Process

Acknowledgement

Some of the slides are adapted from the course "Object-Oriented Analysis and Design Using UML" by IBM Rational .

Objectives

- Explain the six best practices
- Present the (Rational) Unified Process in the context of the six best practices .

Software Development Problems

Symptoms

- User requirements are not met
- Users have mixed requirements
- Modules cannot integrate
- Difficult to maintain
- Late discovery of errors
- Poor user experience
- Poor performance under load
- Team effort not coordinated
- Build-and-release issues .

Patient's experience of disease

Software Development Problems

Diagnosis

- Insufficient requirements analysis
- Ambiguous interfaces
- Fragile architecture
- Unnecessary complexity
- Undetected inconsistencies
- Poor testing
- Subjective assessment
- Waterfall development
- Uncontrolled change
- Insufficient automation .

Root causes identified by doctor

Best Practices to Cure Root Causes

Symptoms	*Root Causes*	*Best Practices*
Needs are not met	Insufficient analysis	Develop iteratively
Mixed requirements	Ambiguous interfaces	Manage requirements
Modules can't fit	Fragile architecture	Use component architecture
Hard to maintain	Unnecessary complexity	Model visually
Late discovery	Undetected inconsistencies	Continuously verify quality
Poor experience	Poor testing	Manage change
Poor performance	Subjective assessment	
Team effort is not coordinated	Waterfall development	
Build-and-release	Uncontrolled change	
	Insufficient automation	

Best Practices

- Develop iteratively
- Manage requirements
- Use component architecture
- Model visually
- Continuously verify quality
- Manage change

Harvested from ...

- Thousands of customers
- Thousands of projects
- Industry experts .

Best Practices

- ***Develop iteratively***
- Manage requirements
- Use component architecture
- Model visually
- Continuously verify quality
- Manage change .

Waterfall Development

What Fundamentals are Important in SE?

What are the Tasks in SE?

- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

Waterfall Development
Characteristics

What Fundamentals are Important in SE?
What are the Tasks in SE?

- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

- Delays handling of critical subsystems
  - All subsystems are treated equal .

Waterfall Development
Characteristics

What Fundamentals are Important in SE?
What are the Tasks in SE?

- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

- Cannot predict time-to-completion
  - 50% of the project completed here? .

Waterfall Development
Characteristics

What Fundamentals are Important in SE?
What are the Tasks in SE?

- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

- Disallows early deployment
  - How much of the system can be run here? .

Waterfall Development
Characteristics

What Fundamentals are Important in SE?
What are the Tasks in SE?

- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

- Delays integration and testing
  - Most difficult phase here .

Waterfall Development

Characteristics

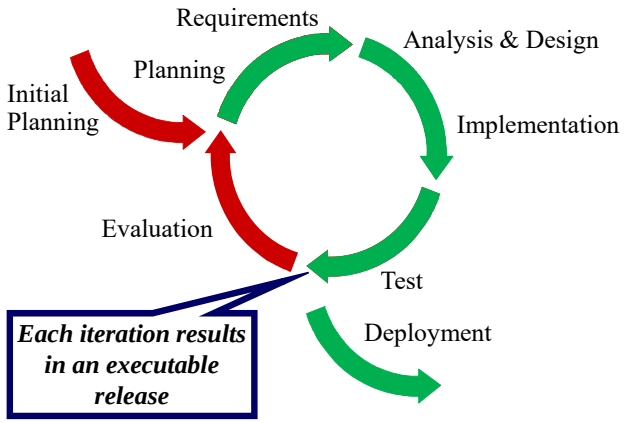
What Fundamentals are Important in SE?

What are the Tasks in SE?

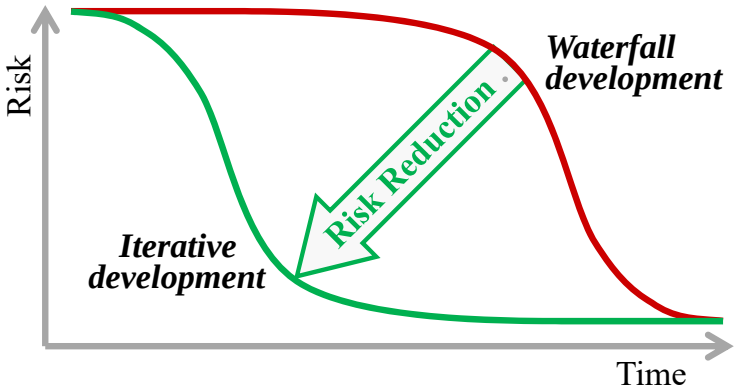
- Feasibility Study
- Requirements Analysis
- Design
- Programming
- Unit test
- Integration test
- User training
- Deployment
- Maintenance

- Frequently results in major unplanned iterations
  - What if a requirements fault is found here?

Iterative Development



Risk Profiles

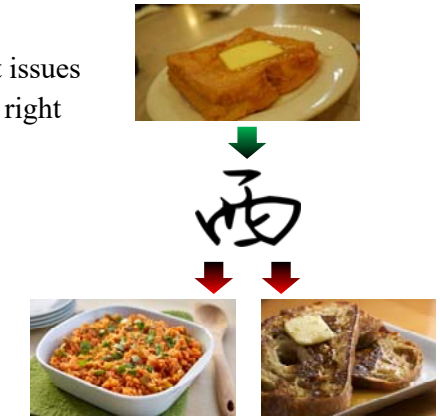


Best Practices

- Develop iteratively
- ***Manage requirements***
- Use component architecture
- Model visually
- Continuously verify quality
- Manage change

Requirements Management

- Make sure you
  - Address the right issues
  - Build the system right



Requirements Management

- Need a systematic approach to
  - capturing
  - organizing
  - documenting
  - and managing

the changing requirements of a software application

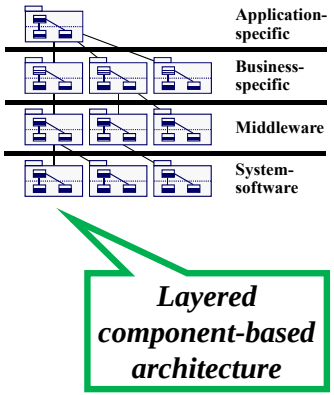
Best Practices

- Develop iteratively
- Manage requirements
- ***Use component architecture***
- Model visually
- Continuously verify quality
- Manage change .

Component-Based Architecture

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

Component-Based Architecture



- Basis for reuse
  - Component reuse
  - Architecture reuse
- Basis for project management
  - Planning
  - Staffing
  - Delivery
- Intellectual control
  - Manage complexity
  - Maintain integrity .

Resilient Architecture

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

Best Practices

- Develop iteratively
- Manage requirements
- Use component architecture
- ***Model visually***
- Continuously verify quality
- Manage change .

Common Types of Specification Languages

- Textual
- Formal
- Graphic .

Example of Textual Specification

McDull https://www.youtube.com/watch?v=0oCP7nZhZJo

麥兜與雞　　曲：舒伯特 詞：謝立文 編：何崇志

我個名叫麥兜兜，我啊媽叫麥太太，
我最喜愛食麥甩咯，一起吃雞一起在歌唱。
我個名叫麥兜兜，我老師叫 Miss Chan Chan，
我最喜愛食碟雞飯，一起吃雞一起在歌唱。
但現實就似一只鴨，吓吓一定要 duck duck。
唔得！唔得！點算嘞？點樣令只雞變做鴨？
含住個雞包仔，望住四寶雞扎，
可嘆現實系要一只鴨，加塊荔芋共我一起扎。
我最喜愛食啫啫雞，我最喜愛食雞 pat pat，
我最喜愛食豉油皇雞翼，一起吃雞一起在歌唱。
我最想吃雞，我最終變臘鴨！鴨！鴨！鴨！鴨！

Textual Specification

Problems:

- Structures are implicit and obscured
  - Lists, tables and hypertext are partial solutions to this problem
- Natural language is prone to ambiguity
  - Unless expressed as long and complex sequences of text, as in legal documents

Hence, only plays supplementary role in analysis and design .

Formal Specification

- Mathematically defined syntax and semantics
- Helps to reason on the problems and solutions
- Helps to verify against ambiguities, inconsistencies, and incompleteness .

Example of Formal Specification

$$
\begin{aligned}
&NEW\_MACHINE \\
&= (coinslot.\$5 \\
&\quad \rightarrow (coinslot.\$5 \rightarrow hatch.cappucino \rightarrow NEW\_MACHINE \\
&\qquad \square\ drink\_button.press \rightarrow hatch.tea \rightarrow NEW\_MACHINE) \\
&\quad \mid coinslot.\$10 \\
&\quad \rightarrow (change\_button.press \rightarrow change.\$5 \rightarrow hatch.tea \\
&\qquad \rightarrow NEW\_MACHINE \\
&\qquad \square\ drink\_button.press \rightarrow hatch.cappucino \\
&\qquad \rightarrow NEW\_MACHINE))
\end{aligned}
$$

Formal Specification

Problems:

- Involve unfamiliar concepts and complex notation
- Difficult in large-scale systems

Use only when necessary, such as

- Defense systems
- Safety critical systems
- Situations where other specifications do not work .

Example of Graphic Specification

McDull

Graphic Specification

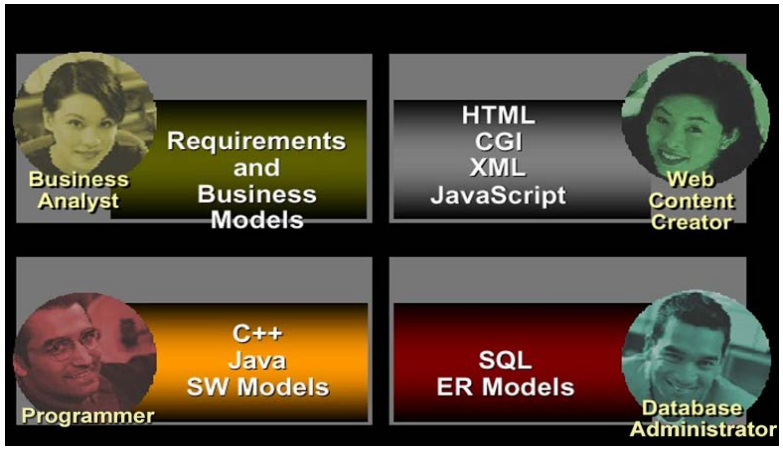
- Capture the structure and behavior of architectures and components
- Easily show how all the pieces fit together
- Help maintain consistency between design and implementation
- Promote human communication
- Hide or expose details as appropriate
- Most popular in analysis and design .

Graphic Specification

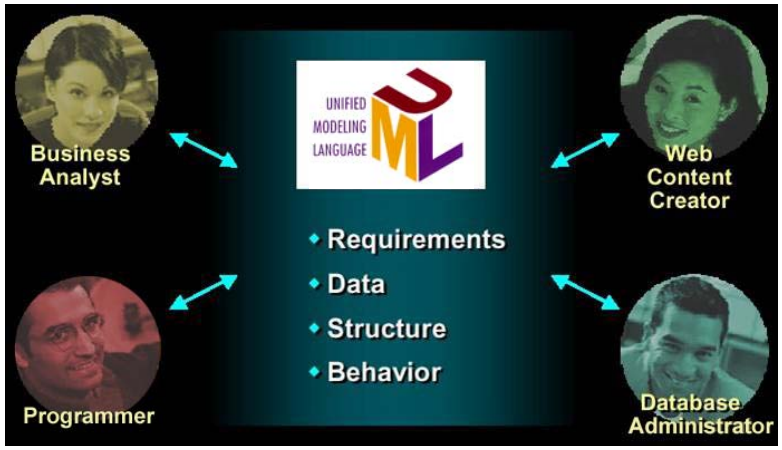
Problems:

- Lack of precise syntax and semantics
- May be interpreted differently by different designers and users .

Multiple Languages = Communication Barriers



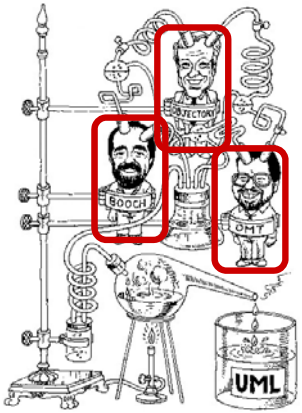
UML: One Language for All Practitioners



History of UML

Integration of

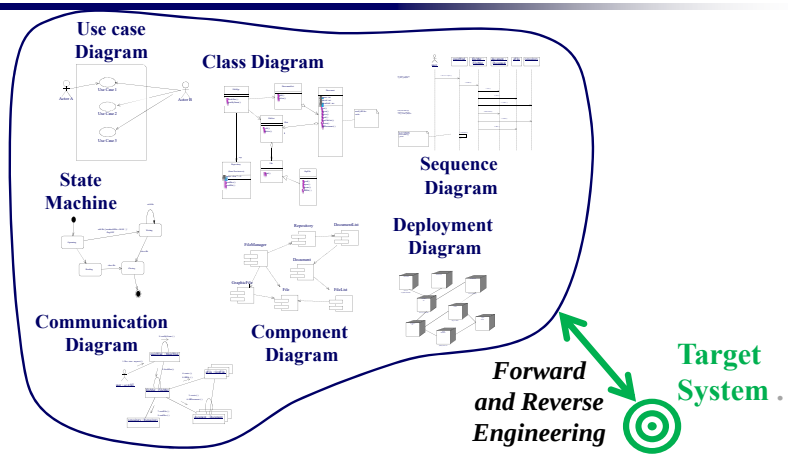
- Object Modelling Technique
  - by James Rumbaugh
- Objectory Process
  - by Ivar Jacobson
- Booch Method
  - by Grady Booch .



Visual Modelling with UML

- Multiple views
- "Semi-formal" syntax and semantics .

Visual Modelling with UML



Best Practices

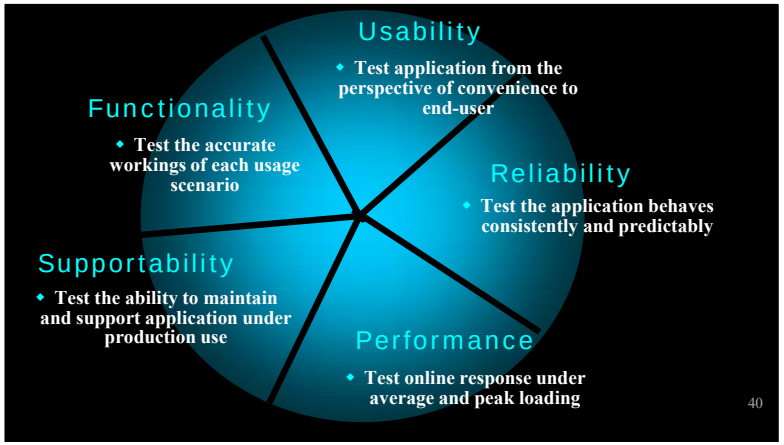
- Develop iteratively
- Manage requirements
- Use component architecture
- Model visually
- ***Continuously verify quality***
- Manage change .

Continuously Verify Software Quality

Earlier detection and repair

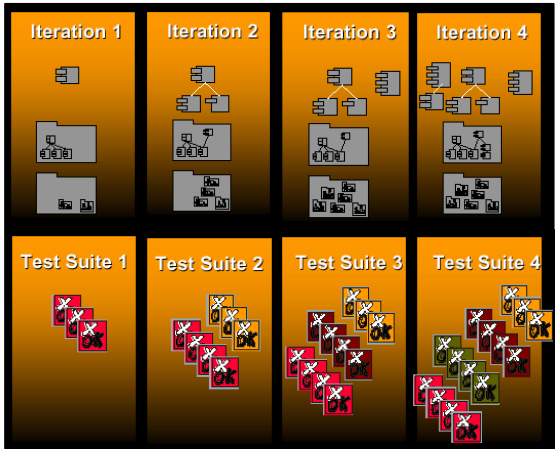
- Problems found earlier are less costly to repair
- Fixing problems earlier leads to higher quality software
- Identifying and resolving problems earlier result in more realistic and reliable development schedule .

Test All Dimensions of Software Quality



Test Each Iteration

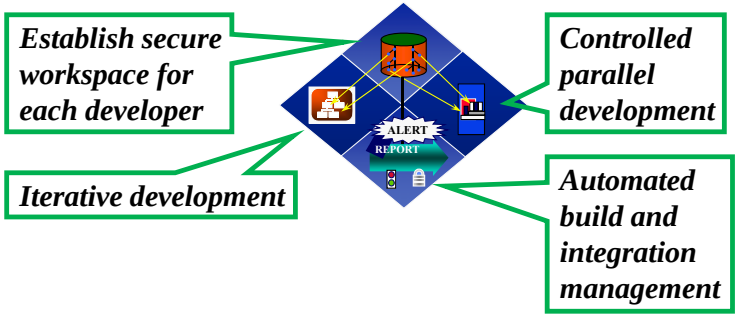
- UML models and implementations
- Tests



Best Practices

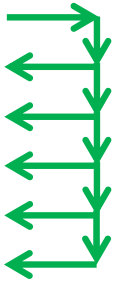
- Develop iteratively
- Manage requirements
- Use component architecture
- Model visually
- Continuously verify quality
- ***Manage change*** .

Change Management for Development Team

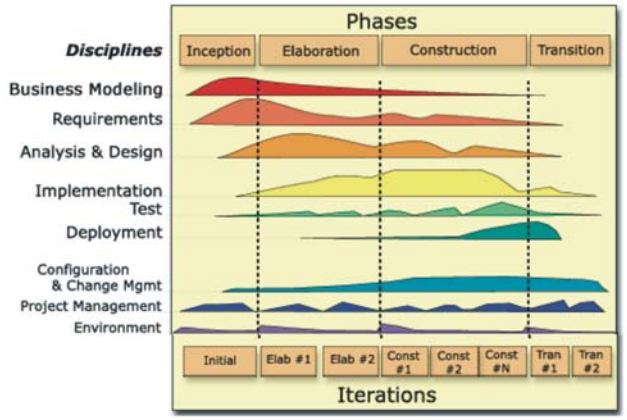


Best Practices Reinforce One Another

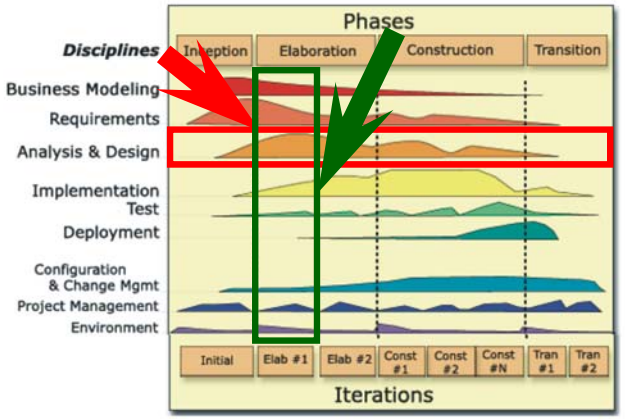
- Develop iteratively
- Manage requirements
- Use component architecture
- Model visually
- Continuously verify quality
- Manage change



Unified Process



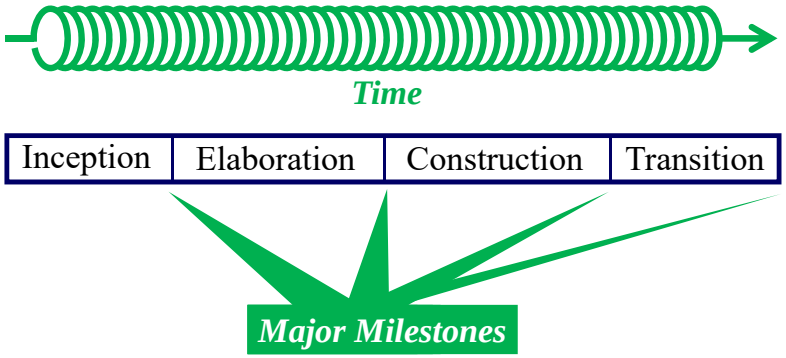
Unified Process



Major Phases in Unified Process

- ***Inception***
  - *Mainly* to specify project scope
- ***Elaboration***
  - *Mainly* to analyse problem domain, plan project, and establish baseline architecture
- ***Construction***
  - *Mainly* to develop system
- ***Transition***
  - *Mainly* to pass final system to users .

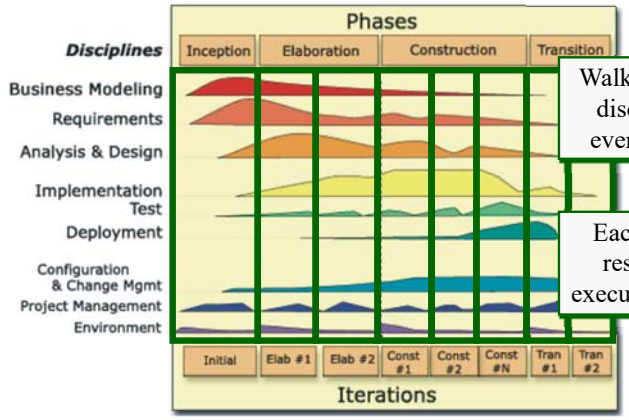
Major Phases in Unified Process



Bringing it All Together:

The Iterative Approach

Disciplines group activities logically



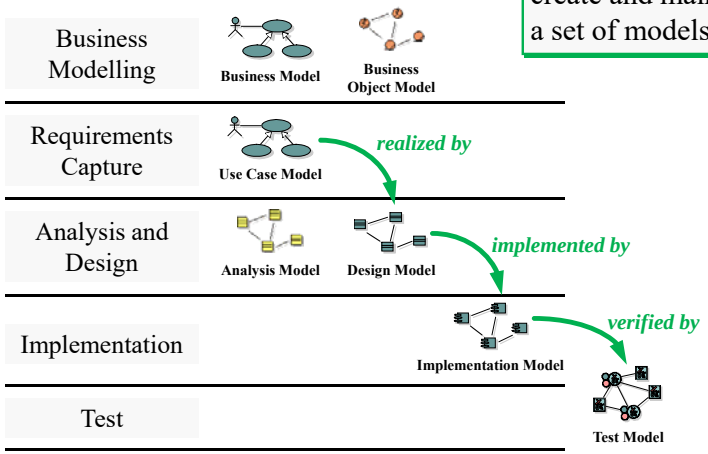
Walk through all disciplines in every iteration

Each iteration results in an executable release

Within Each Phase

Disciplines and Models

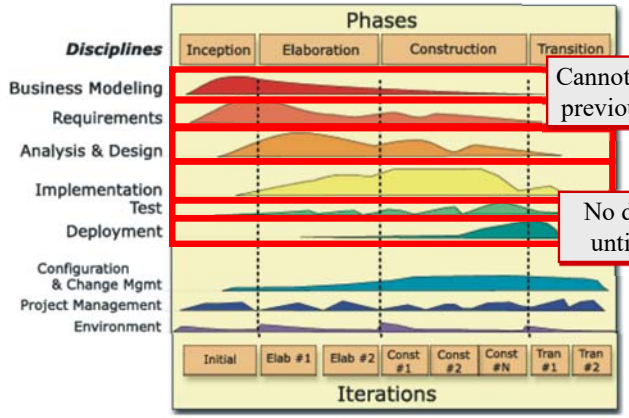
Each discipline describes how to create and maintain a set of models



Compared with

Waterfall Development

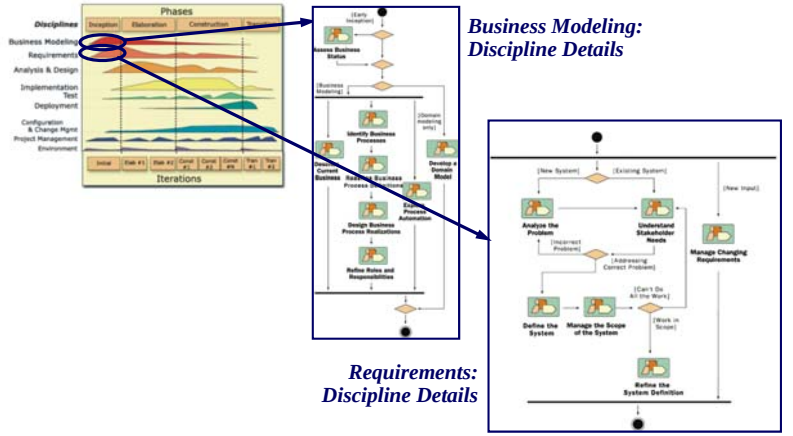
Finish each discipline before the next



Cannot go back to a previous discipline

No deliverables until all done

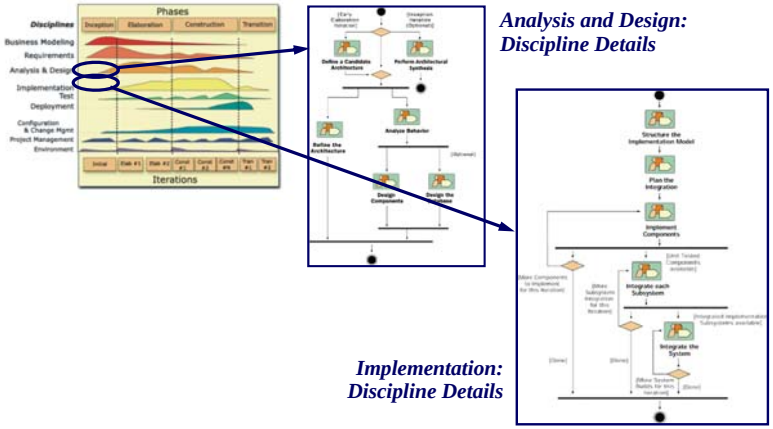
Disciplines Guide Iterative Development



Business Modeling: Discipline Details

Requirements: Discipline Details

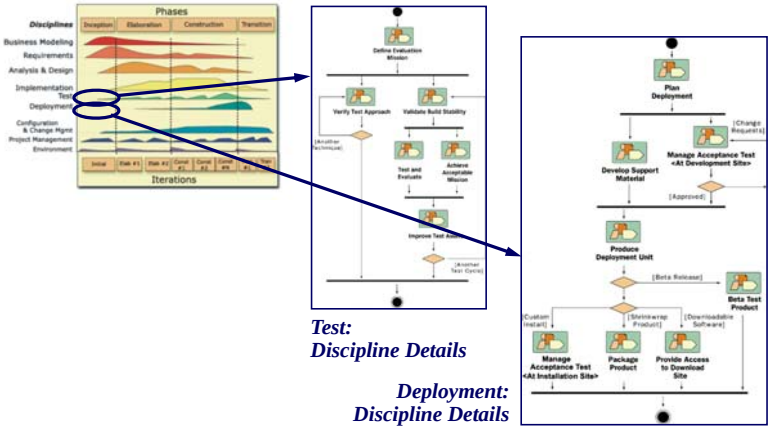
Disciplines Guide Iterative Development



Analysis and Design: Discipline Details

Implementation: Discipline Details

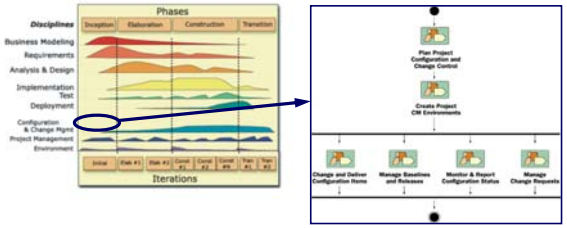
Disciplines Guide Iterative Development



Test: Discipline Details

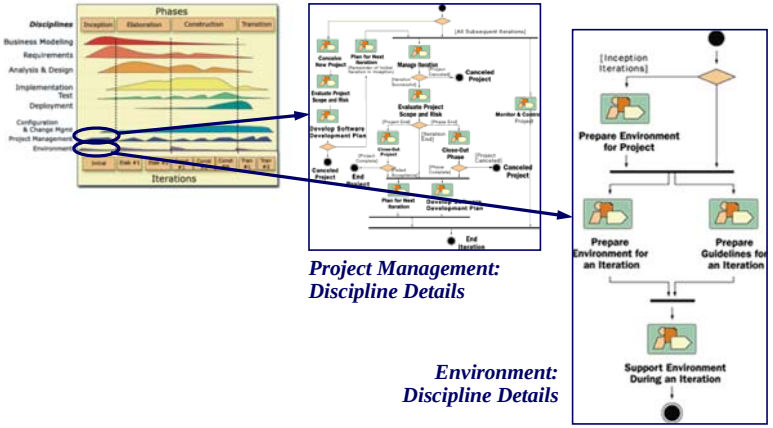
Deployment: Discipline Details

Disciplines Guide Iterative Development



Configuration and Change Management: Discipline Details

Disciplines Guide Iterative Development



Project Management: Discipline Details

Environment: Discipline Details

More Reading Material at Student Request

Develop Iteratively

- Most software teams still use a waterfall process for development projects, completing in strict sequence the phases of requirement analysis, design, implementation, integration and test
- UP represents an iterative approach:
  - It lets you take into account changing requirements
  - Integration is not one "big bang" at the end; instead, elements are integrated progressively
  - With UP, what used to be a lengthy time of uncertainty and pain – taking up to 40% of the total effort at the end of a project – is broken down into six to nine smaller integrations involving fewer elements .

More Reading Material

Develop Iteratively

- Risks are usually discovered or addressed during integration. With the iterative approach, you can mitigate risks earlier.
- Iterative development provides management with a means of making tactical changes to the product – to compete with existing products, for example. It allows you to release a product early with reduced functionality to counter a move by a competitor, or to adopt another vendor for a given technology.
- Iteration facilitates reuse; it is easier to identify reusable components as they are partially designed or implemented than to recognize them during planning
- When you can correct errors over several iterations, the result is a more robust architecture .

More Reading Material

Develop Iteratively

- Project managers often resist the iterative approach, seeing it as a kind of endless and uncontrolled hacking
- In UP, the iterative approach is very controlled; the number, duration, and objectives of iterations are carefully planned, and the tasks and responsibilities of participants are well defined .

More Reading Material

Manage Requirements

- UP is a use-case-driven approach
  - The use cases defined for the system can serve as the foundation for the rest of the development process
  - Use cases used for capturing requirements play a major role in several of the process disciplines, especially design, test, user-interface design, and project management
  - They are also critical to business modeling
- Use cases provide a consistent, visible thread through the system when it performs certain tasks .

More Reading Material

Manage Requirements

- They provide an important link between system requirements and other development artifacts, such as design and tests
- Other object-oriented methods provide use-case-like representation but use different names for it, such as scenarios or threads .

More Reading Material

Use Component Architectures

- Use cases drive UP throughout the entire lifecycle, but design activities center on architecture
- The main focus of early iterations is to produce and validate a software architecture
- In the initial development cycle, this takes the form of an executable architectural prototype that gradually evolves, through subsequent iterations, into the final system .

More Reading Material

Use Component Architectures

- A component can be defined as a nontrivial piece of software: a module, package, or subsystem that fulfills a clear function, has a clear boundary, and can be integrated into a well-defined architecture
  - Physical realization of abstraction in your design
- In defining a modular architecture, you identify, isolate, design, develop, and test well-formed components. These components can be individually tested and gradually integrated to form the whole system.
  - Exploiting the concepts of modularity and encapsulation .

More Reading Material

Use Component Architectures

- Some of these components can be developed to be reusable, especially components that provide solutions to a wide range of common problems
- The advent of commercially successful infrastructures supporting the concept of software components – such as Common Object Request Broker Architecture (CORBA), the Internet, and JavaBeans – has launched a whole industry of off-the-shelf components for various domains, allowing developers to buy and integrate components rather than develop them in-house
  - Shift software development from programming software (one line at a time) to composing software (by assembling components) .

More Reading Material

Use Component Architectures

- UP supports component-based development in several ways
  - The iterative approach allows developers to progressively identify components and decide which ones to develop, which ones to reuse, and which ones to buy
  - The architecture defines the components and the ways they integrate, as well as the fundamental mechanisms and patterns by which they interact
  - Concepts such as packages, subsystems, and layers are used during analysis and design to organize components and specify interfaces
  - Testing is organized around single components first and then is gradually expanded to include larger sets of integrated components .

More Reading Material

Model Visually

- Models are simplifications of reality; they help us to understand and shape both a problem and its solution, and to comprehend large, complex systems that we could not otherwise understand as a whole
- A large part of UP is about developing and maintaining models of the system under development
- The Unified Modeling Language (UML) is a Graphic language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system .

More Reading Material

Model Visually

- UML does not tell you how to develop software
- UP is a guide to the effective use of UML for modeling
  - It describes the models you need, why you need them, and how to construct them .

More Reading Material

Continuously Verify Quality

- Our concern about quality is focused on two areas: product quality and process quality
- Product quality
  - The quality of the principal product being produced (the software or system) and all the elements it comprises (for example, components, subsystems, architecture, and so on) .

More Reading Material

Continuously Verify Quality

- Process quality
  - The degree to which an acceptable process (including measurements and criteria for quality) was implemented and adhered to during the manufacturing of the product
  - Additionally, process quality is also concerned with the quality of the artifacts (such as iteration plans, test plans, use-case realizations, design model, and so on) produced in support of the principal product .

More Reading Material

Manage Change

- By allowing flexibility in the planning and execution of the development and by allowing the requirements to evolve, iterative development emphasizes the vital issues of keeping track of changes and ensuring that everything and everyone is in sync
- Focused closely on the needs of the development organization, change management is a systematic approach to managing changes in requirements, design, and implementation .

More Reading Material

Manage Change

- It also covers the important activities of keeping track of defects, misunderstandings, and project commitments as well as associating these activities with specific artifacts and releases
- Change management is tied to configuration management and measurements .

More Reading Material

Inception Phase

- ***Purpose***
  - To establish the business case for a new system or for a major update of an existing system
  - To specify the project scope
- ***Outcome***
  - A general vision of the project's requirements, that is, the core requirements
    - Initial use-case model and domain model (10–20% complete)
  - An initial business case, including:
    - Success criteria (such as revenue projection)
    - An initial risk assessment
    - An estimate of resources required
- ***Milestone:*** Lifecycle objectives .

More Reading Material

Elaboration Phase

- ***Purpose***
  - To analyze the problem domain
  - To establish a sound architectural foundation
  - To address the highest risk elements of the project
  - To develop a comprehensive plan showing how the project will be completed
- ***Outcome***
  - Use-case and domain model 80% complete
  - An executable architecture and accompanying documentation
  - A revised business case, including revised risk assessment
  - A development plan for the overall project
- ***Milestone:*** Lifecycle architecture .

More Reading Material

Construction Phase

- ***Purpose***
  - To incrementally develop a complete software product which is ready to transition into the user community
- ***Products***
  - A complete use-case and design model
  - Executable releases of increasing functionality
  - User documentation
  - Deployment documentation
  - Evaluation criteria for each iteration
  - Release descriptions, including quality assurance results
  - Updated development plan
- ***Milestone:*** Initial operational capability .

More Reading Material

Transition Phase

- ***Purpose***
  - To transition the software product into the user community
- ***Products***
  - Executable releases
  - Updated system models
  - Evaluation criteria for each iteration
  - Release descriptions, including quality assurance results
  - Updated user manuals
  - Updated deployment documentation
  - “Post-mortem” analysis of project performance
- ***Milestone:*** Product release .