# *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 .

***Systems Analysis from user requirements***



*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 .

***Systems Design by innovation and invention***

## 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 .

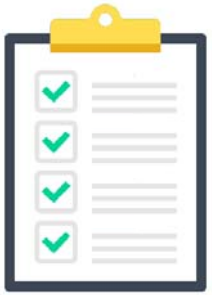


## Best Practices

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



## Waterfall Model



- ***Traditional development process***
- ***Strictly analysis followed by design***
- ***No iteration allowed*** .

# Waterfall Model

*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?*

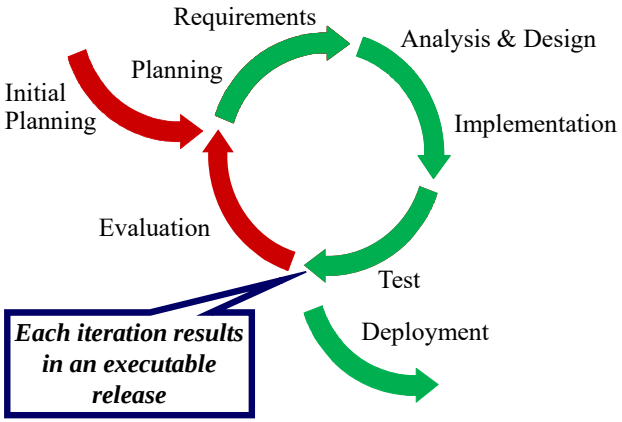
### What are the Tasks in SE?

- 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? .

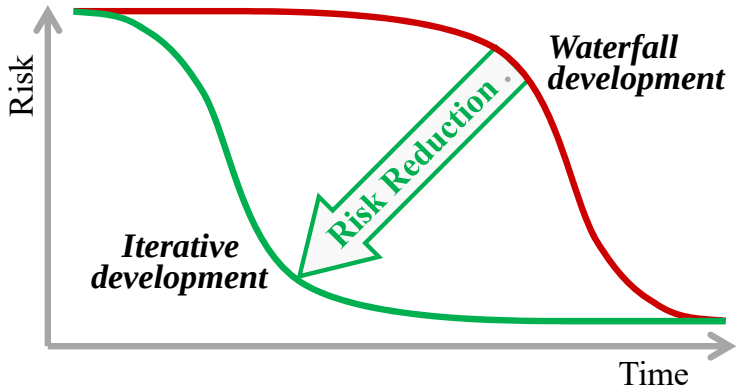


# Iterative Development






# 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

***Systems Analysis from user requirements***

## Requirements Management

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

***Systems Design by innovation and invention***

*Requirements Management*

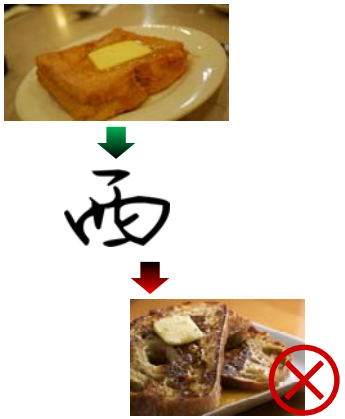
## Examples

- ***Make sure you***
  - Address the right issues

*Requirements Management*

## Examples

- *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 user requirements of software application***

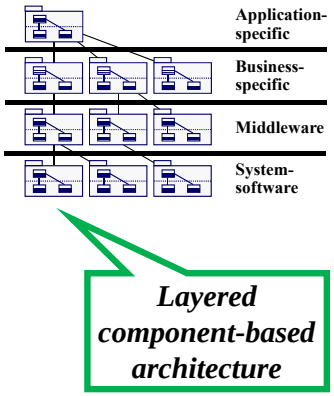
## Best Practices

- 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

*NEW_MACHINE*
= ( *coinslot*.$5
→ ( *coinslot*.$5 → *hatch.cappucino* → *NEW_MACHINE*
□ *drink_button.press* → *hatch.tea* → *NEW_MACHINE* )
| *coinslot*.$10
→ ( *change_button.press* → *change*.$5 → *hatch.tea*
→ *NEW_MACHINE*
□ *drink_button.press* → *hatch.cappucino*
→ *NEW_MACHINE* ) )

## 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 .

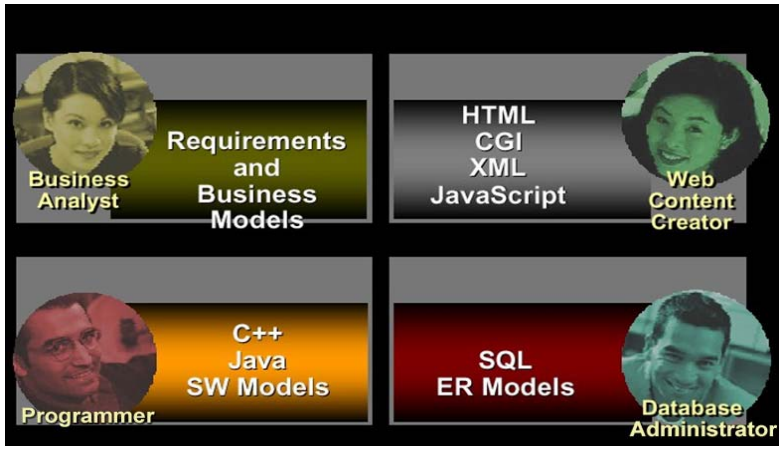


## Graphic Specification

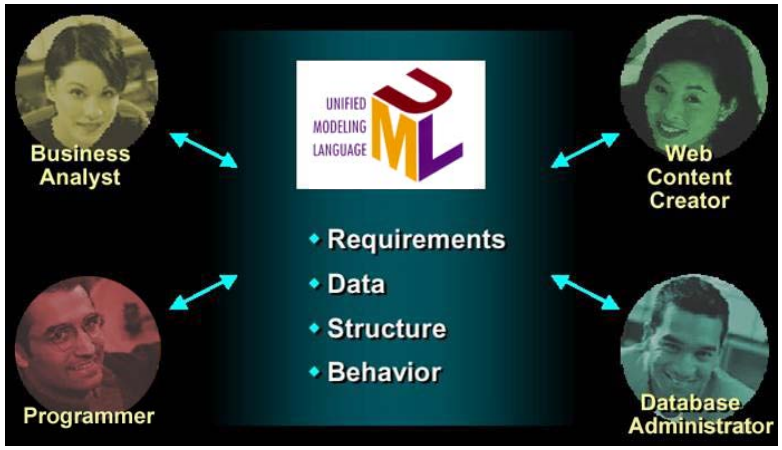
***Problems:***

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

## Multiple Languages = Communication Barriers



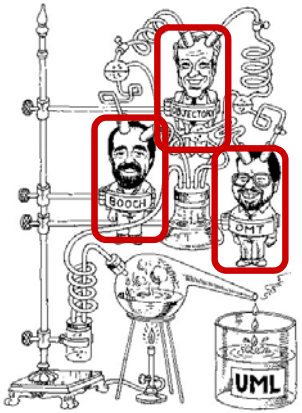

## UML: One Language for All Practitioners




## History of UML

***Integration of***

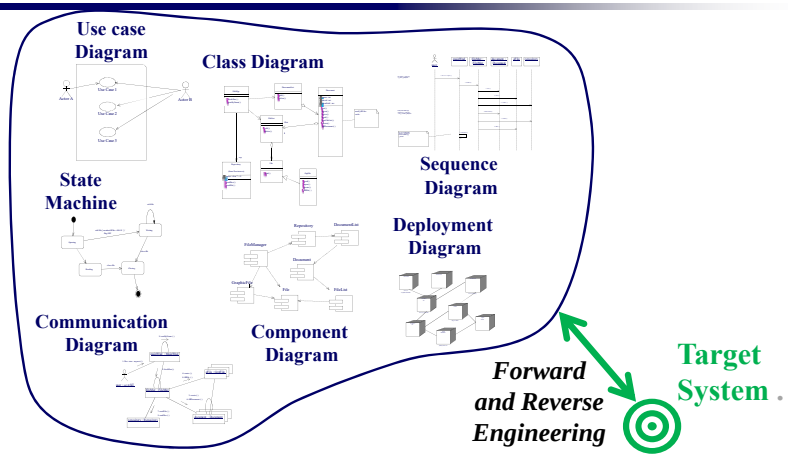
- Object Modelling Technique
  - by James Rumbaugh
- Objectory Process
  - by Ivar Jacobson
- Booch Method
  - by Grady Booch .




## 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 .

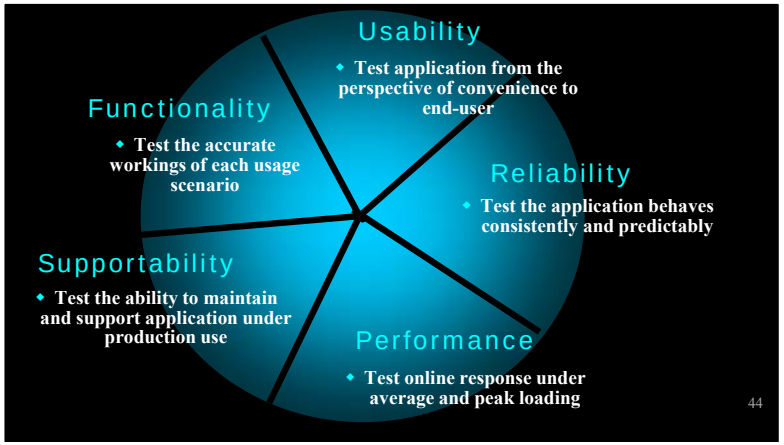


# Continuously Verify Software Quality


***Earlier detection and repair***

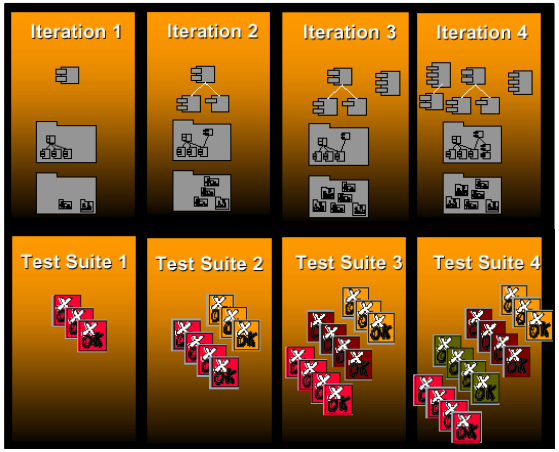
- Problems found earlier are less costly to repair
- Fixing problems earlier leads to higher quality software
- Identifying and resolving problems earlier result in more realistic and reliable development schedule .

# Test All Dimensions of Software Quality



## Test Each Iteration

- UML models and implementations
- Tests

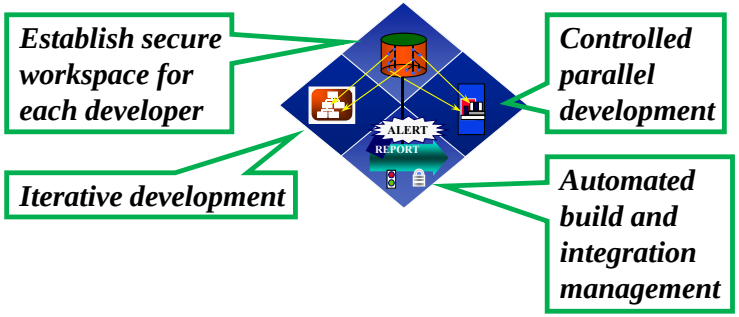



## Best Practices

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



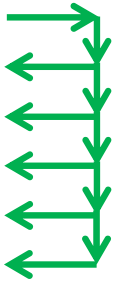
## Change Management for Development Team



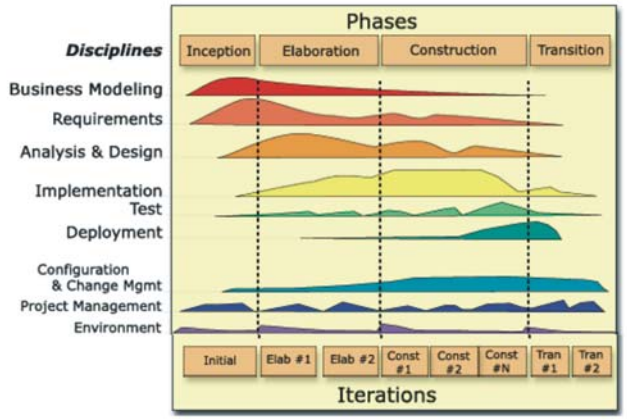



## Best Practices Reinforce One Another

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

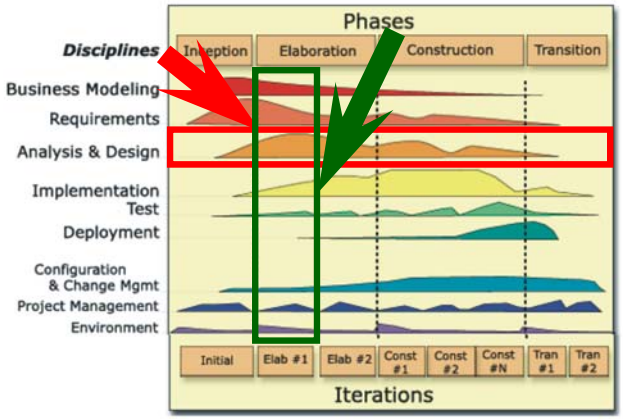


## Unified Process






## Unified Process



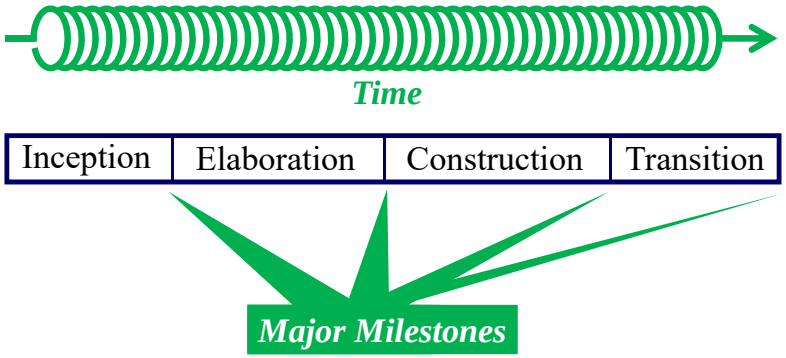



## Major Phases in Unified Process

- ***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 .



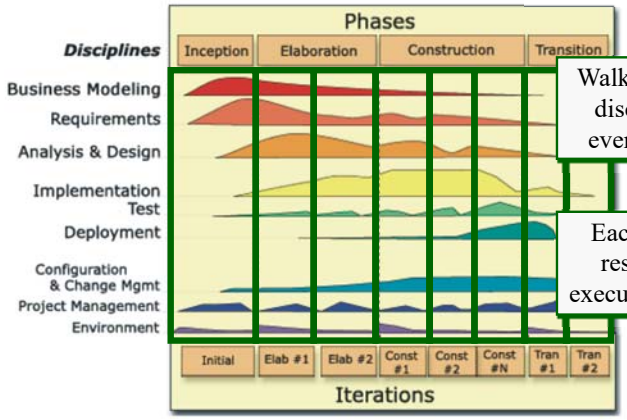
## Major Phases in Unified Process



*Bringing it All Together:*

## The Iterative Approach

Disciplines group activities logically




Walk through all disciplines in every iteration

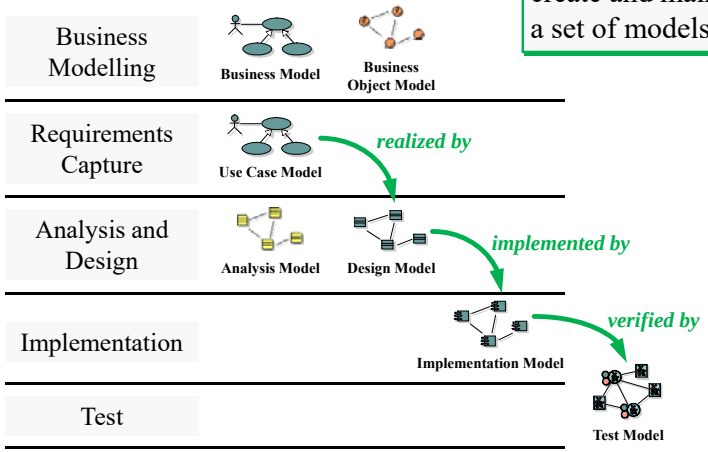
Each iteration results in an executable release



*Within Each Phase*

## Disciplines and Models

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

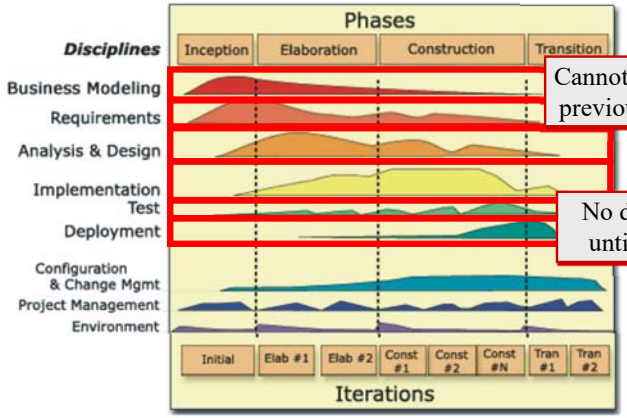





*Compared with*

## Waterfall Development

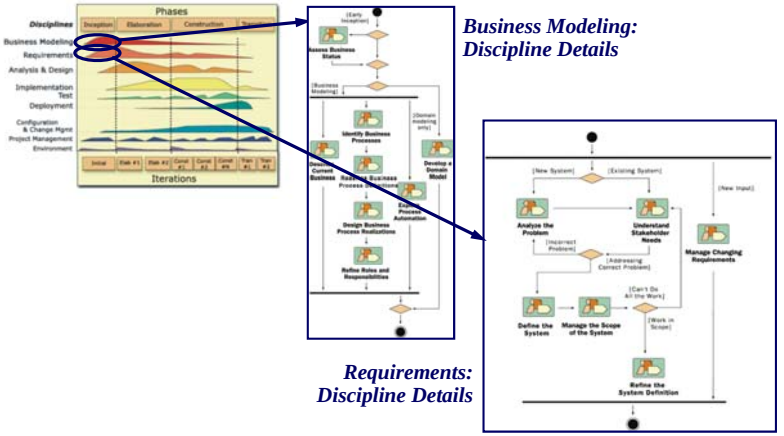
Finish each discipline before the next




Cannot go back to a previous discipline

No deliverables until all done

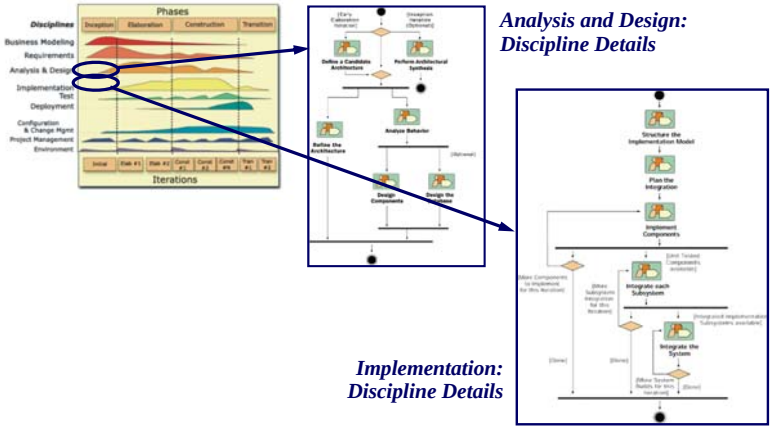
## Disciplines Guide Iterative Development



*Business Modeling: Discipline Details*

*Requirements: Discipline Details*

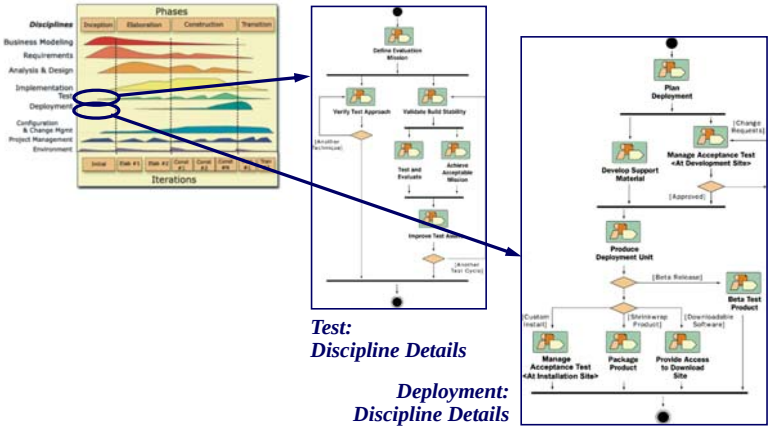
# Disciplines Guide Iterative Development




*Analysis and Design: Discipline Details*

*Implementation: Discipline Details*

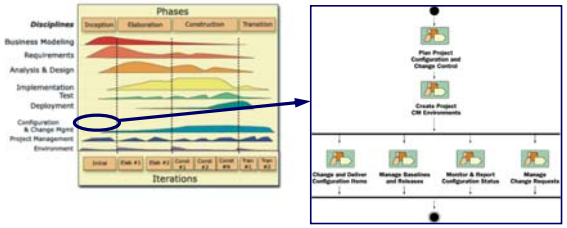
# Disciplines Guide Iterative Development




*Test: Discipline Details*

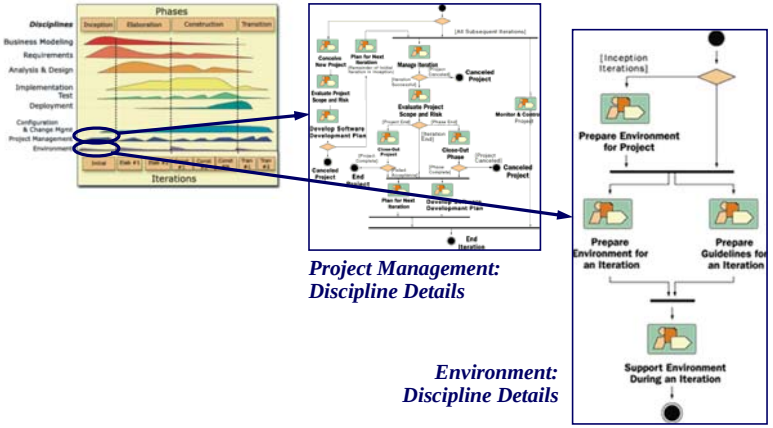
*Deployment: Discipline Details*

# Disciplines Guide Iterative Development




*Configuration and Change Management: Discipline Details*

# Disciplines Guide Iterative Development




*Project Management: Discipline Details*

*Environment: Discipline Details*