Requirements

Develop A Vision

1. Gain agreement on the problem to solve
2. Identify stakeholders
3. Define system features and function
4. Define system boundaries
5. Identify constraints on the system

Develop Requirements

- Should we try to specify all the requirements at beginning of a project?
- Why?

Problem with early complete req'ts

In a study of projects that use waterfall style, complete requirements they found...

- what % of the features were never used?
- what % of the features were used rarely?

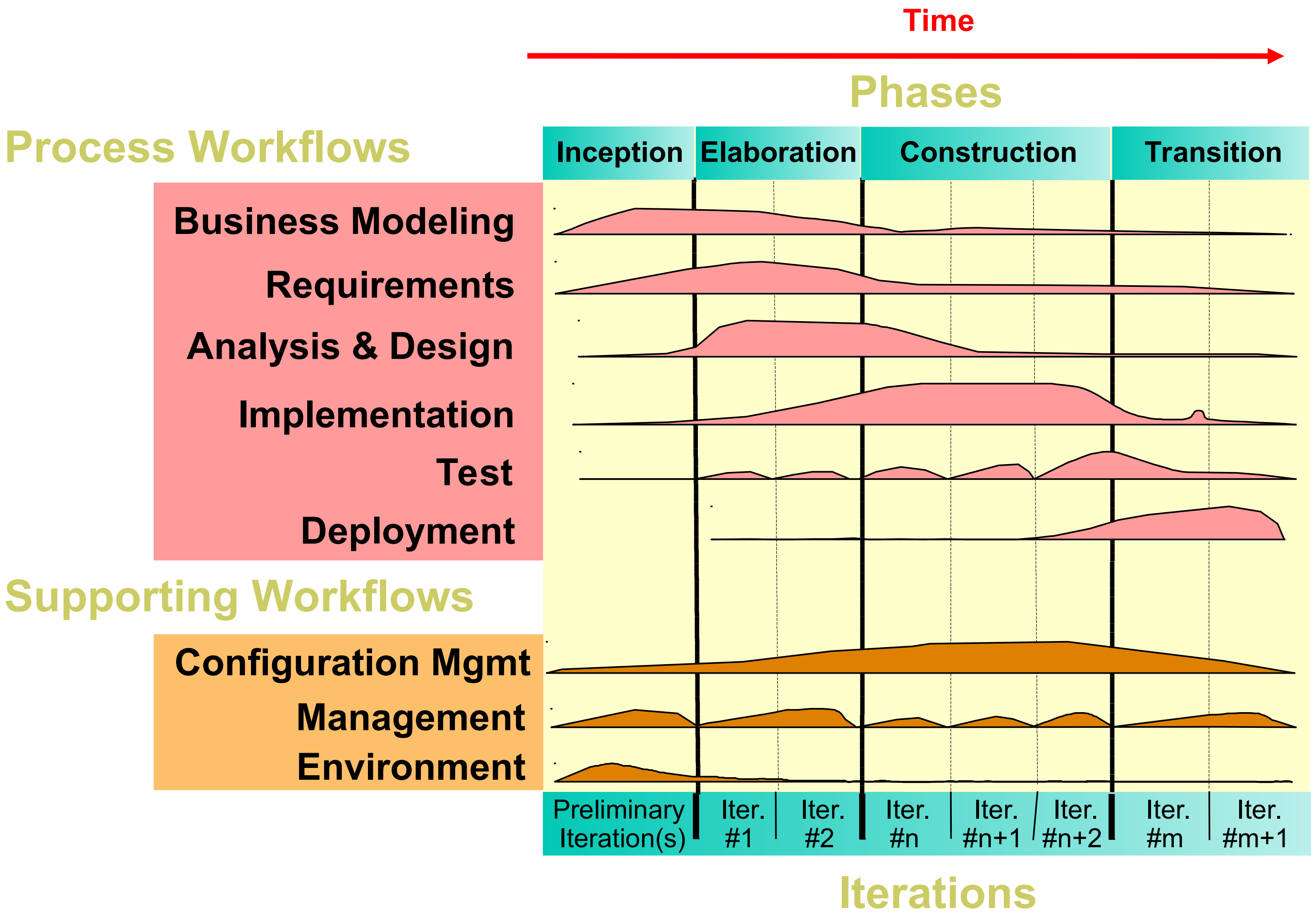
Changing Requirements

- On average, what % of the requirements will change?

When to do requirements in UP?

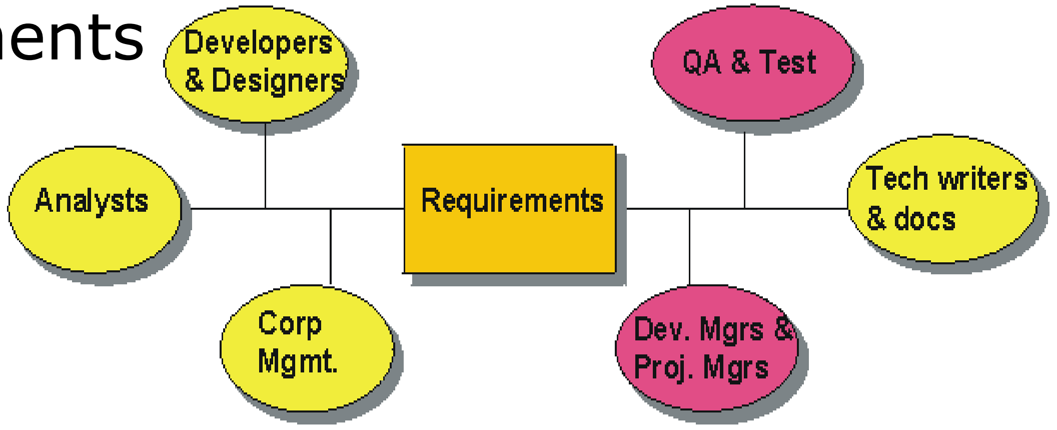
- In the UP, in what phase(s) are requirements gathered and written?

Do you remember this?



Important Requirement Concepts

- Types of requirements (FURPS+)
- Characteristics of good requirements
- How to elicit
- How to document
- Verify requirements
- Manage requirements
- Tracability



Requirements Goals: CMMI Guidelines

SG 1: Develop Customer Requirements

- Stakeholder needs, expectations, constraints, and interfaces are collected and translated into customer requirements.

SG 2: Develop Product Requirements

- Customer requirements are refined and elaborated to develop product and product-component requirements.

Requirements Goals: CMMI Guidelines

SG 3: Analyze and Validate Requirements

- The requirements are analyzed and validated, and a definition of required functionality is developed.

Types of Requirements

- What does "FURPS" mean?

FURPS

Give an example of each type:

Functional	
Usability	
Reliability aka "Dependability"	
Performance	
Supportability	

FURPS

Give an example of each type:

Functional	Create sales report Process sale
Usability	Display should be readable at a distance of 1 meter
Reliability aka "Dependability"	The application should never crash
Performance	Returns reply to query within 1 second 99% of the time under normal load.
Supportability	Can be used on any computer supported by Java SE 6 or newer with at least 1GB RAM.

FURPS+

- What does the "+" mean?

Some "+" requirement types

implementation	language and tools, resource limits
interface	must interface with other systems
operations	managing the software after its installed
packaging	media, box
platform	OS, hardware; can be part of "Supportability".
legal	licensing issues, liability

Larman, section 5.4 and 7.4

Give examples

implementation	
interface	
operations	
packaging	
legal	

Larman, section 5.4

Classify these requirements

- POS should be written in Java.
- The display should be readable by person of normal eyesight at a distance of 1 meter.
- Must be able to process sales and refunds.
- Must be able to control a cash drawer the adheres to JavaPOS standard. The cash drawer manufacturer shall provide a driver written in Java.
- should have GUI management interface for managing user accounts, connection to store's product catalog, and examining logs.

Classify these Requirements

- Must run on Linux supported by Java SE 5.0 or newer.
- If network connection to the store product catalog is down, the POS should still be able to process sales using (a) recently cached product data, and (b) manual key-in of prices by POS operator.

Documenting Requirements

- In what form are they written?

- What documents are related to requirements? (Larman, 5.5)
  - use case model (mostly functional req.)
  - supplementary requirements specification
  - glossary, data dictionary
  - business rules

Eliciting Requirements

What are some techniques for finding requirements?

Eliciting Requirements

Actively Involve Customer & Stakeholders

- Interviews
- Joint Application Development (JAD)
- Brain storming
- Vision

Other techniques

- Observation
- Procedure manuals
- Interfaces - how it connects to other systems

Requirements Activities

From Bruegge (OOSE):

1. identify stakeholders
2. identify scenarios: concrete examples of how the system is used... or will be used
3. learn about existing system -- observation, interview, procedure manuals, workflow
4. develop use cases: consolidate scenarios. Use cases define the scope of the project
5. identify relationships between use cases
6. identify non-functional requirements

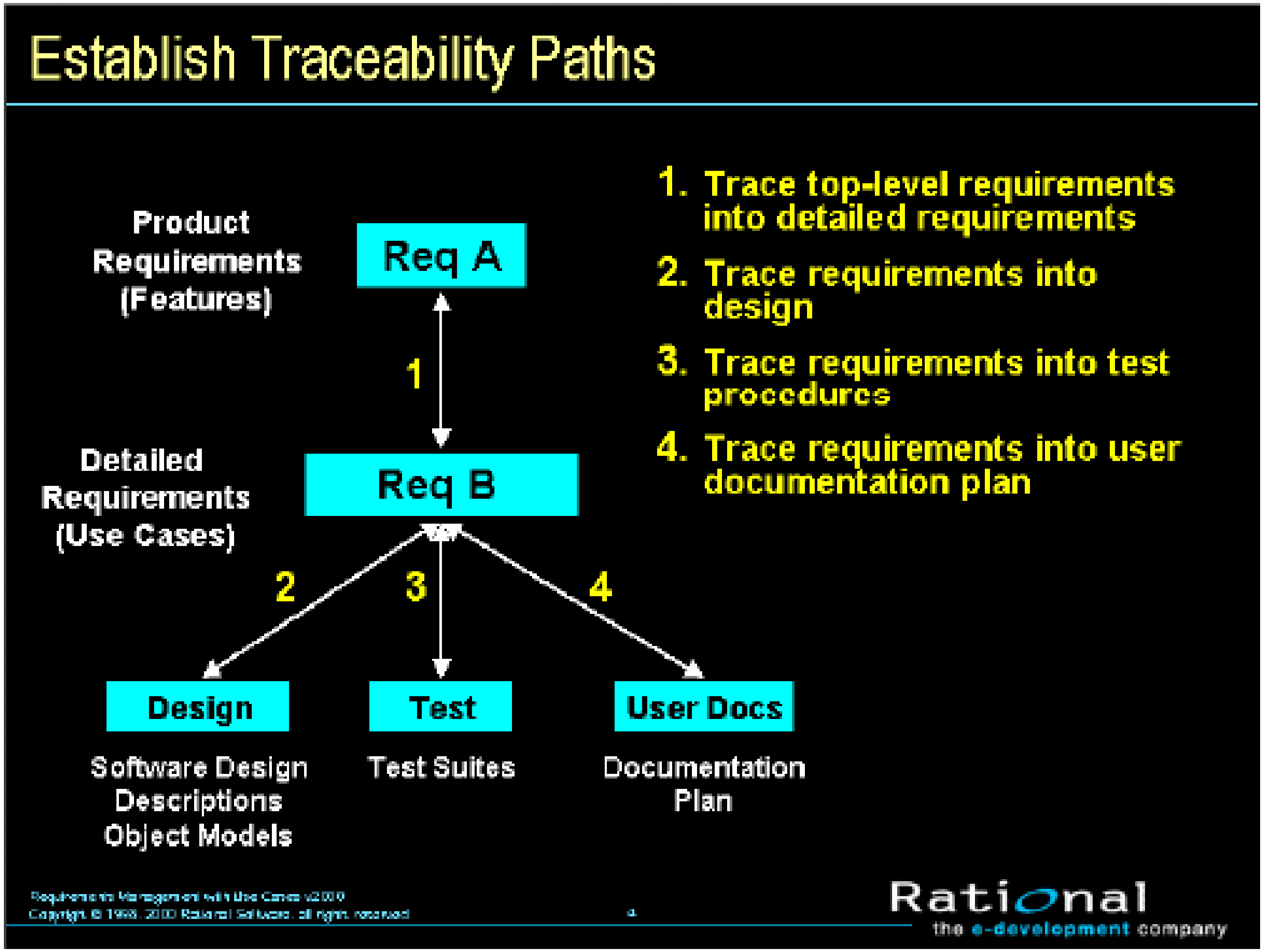
Why Manage Requirements?

- Why?

Manage Requirements

- Agree on *how* and *where* requirements will be written
- Procedure for *how* a requirement is reviewed and accepted
- Procedure for requirement change requests
- The impact of changing a requirement should be studied
  - what work products need to change?
  - what is effect on other requirements?
  - how much will it cost?
- Review impact with stakeholders and get sign-off

Requirement Tracing


Product Requirements (Features)
Req A
1
Detailed Requirements (Use Cases)
Req B
2
3
4
Design
Test
User Docs
Software Design Descriptions Object Models
Test Suites
Documentation Plan
Rational
the e-development company

Establish Traceability Paths

1. Trace top-level requirements into detailed requirements
2. Trace requirements into design
3. Trace requirements into test procedures
4. Trace requirements into user documentation plan

Writing Requirements

- What is a Use Case?
- Why write requirements as Use Cases instead of a list of required features?

Guidelines for Writing Use Cases

- See Larman, chapter 6-7
- See Cockburn, *Writing Effective Use Cases* (ebook), the first few pages.

User Stories

As a [role] [with context]

I want to [goal]

In order to [

Guidelines for Good Requirements

What are some criteria for good requirements?

Good Requirements

- Clear
  - both client and developers can understand
  - use domain terms
- Unambiguous
- Consistent
- Unique - no redundancy
- Feasible
- Testable or verifiable
- Customer needs, not implementation details

What's wrong with these requirements?

- The system should be able to handle many simultaneous users.
- The system should never crash.
- The UI should not be coupled to the Tax Adapter class
- Hardware requirements for server are:
  - Intel Quad Core 2.4 GHz cpu
  - 4GB DDR400 memory
  - minimum 160GB serial ATA disk drive

Advantages of Use Cases

- Client can understand them and contribute
- Requirements are *in context* -- a usage pattern or a story
- helps group related requirements together
- help avoid forgetting requirements
- help identify non-functional requirements
- easier to specify alternatives
- specifying alternative paths help make the system more robust

Use Case Guideline 1

Each step in use case should be:

1. interaction between actor and system, or
2. action performed by system
   - do something that changes state of system
   - verify something

1. cashier enters item code 2. pos displays item description and price	1. cashier needs to enter barcode 2. there are many items in the inventory

Use Case Guideline 2

Use concise wording. Omit words like "the", "a" or verbose descriptions.

1. cashier enters item code	1. The cashier locates the barcode symbol on the item and enters the item's barcode into system
2. pos displays item description and price	2. The POS appends a line showing item description, unit price, and item subtotal

Use Case Guideline 3

Write use cases in terms of *intention*.

Avoid:

- User Interface details
- implement specific wording

1. cashier enters item code	1. cashier scans the stock code
2. cashier authenticates himself to system	2. cashier types in login and password

Use Case Example (MRT Ticket Vending Machine)

Use Case: Purchase ticket

Actor: Passenger

Pre-conditions:

- Passenger is in front of ticket Distributor.
- Passenger has sufficient money (coins) to purchase ticket.

Post-conditions:

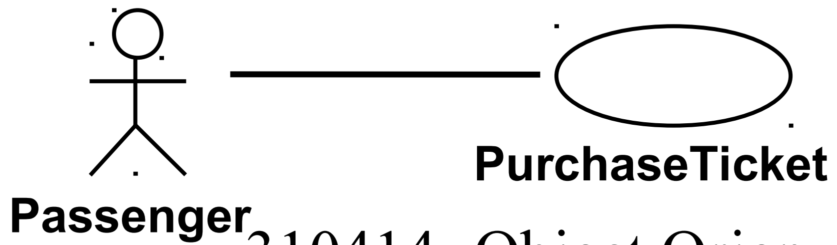
- Passenger has ticket.

Main flow:

1. Passenger inserts coins
2. Distributor displays the possibilities according to the amount deposited
3. Passenger selects one of the options

4. Distributor issues ticket.

5. Distributor returns change.


PurchaseTicket
Passenger