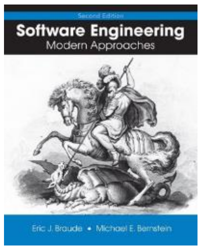
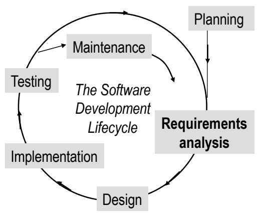
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 10: Principles of Requirements Analysis



Phase most relevant to this chapter is shown in bold

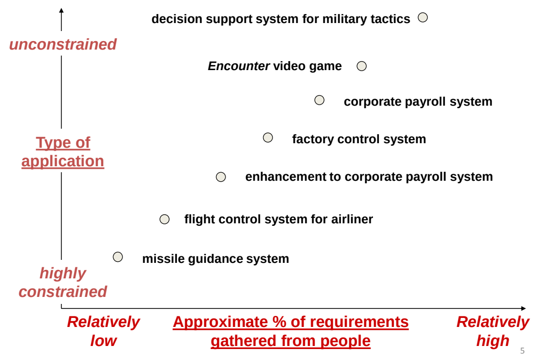
Learning goals of this chapter

- Why the term requirements *analysis*?
- What is the value of writing down requirements?
- Where do requirements come from?
- What is the difference between high-level and detailed requirements?
- What is the difference between functional and non-functional requirements?
- How do you document requirements?
- What does traceability mean?
- How do agile methods handle requirements?
- What are good tips for student project requirements analysis?

The Meaning of *Requirements Analysis*

- The process of understanding what's wanted and needed in an application. For example, you may know that you *want* a colonial house in New England, but you may not know that you will probably *need* a basement for it.
- We express requirements in writing to complete our understanding and to create a contract between developer and customer.

Sources of Requirements: People vs. Other (adapted from Brackett [Br])



High-level *versus* Detailed Requirements

- High-level requirements (market requirements): an overview, which is relatively readable and well suited to customers.
  - Anyone wanting to get an idea of what the application is all about reads the high-level requirements. For instance, considering the video store
  - The Video Store application shall enable clerks to check DVDs in and out.
  - The following shows a sketch of the main user interface: . . .
- Detailed requirements: specially useful for developers, who need to know precisely what they have to build. They also should be understandable to the customer, and should not contain developer jargon where possible.
  - The daily late charge o n a D V D shall b e computed at half the regular two-day rental rate, up to the value of the DVD listed in the "Intergalactic Video Catalog." When the amount owed reaches this value, the total l ate charge is computed as this amount plus $5.
  - When the "comm it" button is pressed on CUI 37, the CUI shall disappear and CUI 15 shall appear with a superimposed green border (RGB = 6, 32, 8) and the name and address fields filled with the data for the customer.

Functional *vesus* Non-Functional Requirements

- Functional requirement: also known as behavioral requirement, specify services that the application must provide. For example:
  - The application allows clerks to check out DVDs
  - The application allows clerks to display the customer's account status
- Non-functional requirement: any requirement that does not specify functionality. It qualifies a service or services. It needs to be specific, quantifiable, and testable.
  - The system shall retrieve user information quickly
    - What does retrieve means? (vague)
    - What does quickly means? (not quantifiable)
  - Once the Ok button is pressed on the "Retrieve account information" screen, the user's account information shall be displayed in less than 3 seconds.

Non-Functional Requirements

- **Quality attributes**
  - **Reliability and Availability** (observed faults, average up time)
  - **Performance** (speed, throughput, storage)
  - **Security** (malicious and non-malicious compromise of data or functionality)
  - **Maintainability** (cost to maintain)
  - **Portability** (move to a different operating enviroment)
- **Constraints** on the application or its development
- **External interfaces** that the application "talks to"
  - **Hardware**
  - **Other software**
  - **Communication with external agents**
- **User interfaces**
- **Error handling**

Examples of Constraints

- **Platform**
  - e.g., the application must execute on any 1GH Linux computer
- **Development Media**
  - e.g., the application must be implemented in Java
  - e.g., Rational Rose must be used for the design

External Interface Requirements

- **Hardware**
  - e.g., "the application must interface with a model 1234 bar code reader"
- **Software**
  - e.g., "the application shall use the company's payroll system to retrieve salary information"
  - e.g., "the application shall use version 1.1 of the Apache server"
- **Communications**
  - e.g., "the application shall communicate with human resources applications via the company intranet"
  - e.g., "the format used to transmit "article expected" messages to cooperating shipping companies shall use XML standard 183.34 published at http://…"

Steps for Constructing User Interfaces[1]

Step 1: Know your user	(H[2])
Step 2: Understand the business function in question	(H)
Step 3: Apply principles of good screen design	(H, D[3])
Step 4: Select the appropriate kind of windows	(H, D)
Step 5: Develop system menus	(H, D)
Step 6: Select the appropriate device-based controls	(H)
Step 7: Choose the appropriate screen-based controls	(H)
Step 8: Organize and lay out windows	(H, D)
Step 9: Choose appropriate colors	(D)
Step 10: Create meaningful icons	(H, D)
Step 11: Provide effective message, feedback, & guidance	(D)

[1] adapted from Galitz [2]a high-level requirement process [3]a detailed requirement process

Examples of Error Handling Options

- Ignore
- Warn user
- Allow unlimited retries
- Log and proceed anyway
- Substitute default values
- Shut down

IEEE 830-1998 SRS Table of Contents

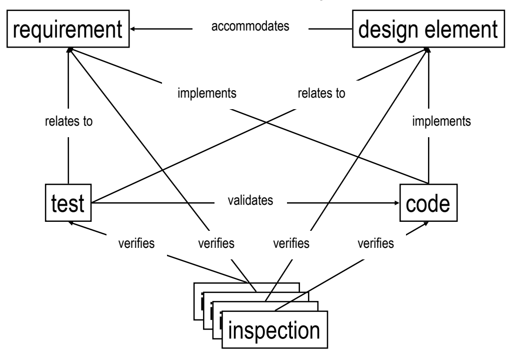
1. Introduction
1.1. Purpose
1.2. Scope
1.3. Definitions, acronyms & abbreviations
1.4. References
1.5. Overview
2. Overall description
2.1. Product perspective
2.1.1. System interfaces
2.1.2. User interfaces
2.1.3. Hardware interfaces
2.1.4. Software interfaces
2.1.5. Communications interfaces
2.1.6. Memory constraints
2.1.7. Operations
2.1.8. Site adaptation requirements
2.2. Product functions
2.3. User characteristics
2.4. Constraints
2.5. Assumptions and dependencies
2.6 Apportioning of requirements
3. Specific requirements
-- see Figure 9
Appendixes

Replaced by ISO/IEC/IEEE 29148:2011

IEEE 830-1998 Detailed Requirements

3.1 External interfaces
3.2 Functional requirements
-- organized by feature, object, user class, etc.
3.3 Performance requirements
3.4 Logical database requirements
3.5 Design constraints
3.5.1 Standards compliance
3.6 Software system attributes
3.6.1 Reliability
3.6.2 Availability
3.6.3 Security
3.6.4 Maintainability
3.6.5 Portability
3.7 Organizing the specific requirements
3.7.1 System mode -- or
3.7.2 User class -- or
3.7.3 Objects (see right) -- or
3.7.4 Feature -- or
3.7.5 Stimulus -- or
3.7.6 Response -- or
3.7.7 Functional hierarchy -- or
3.8 Additional comments

Traceability



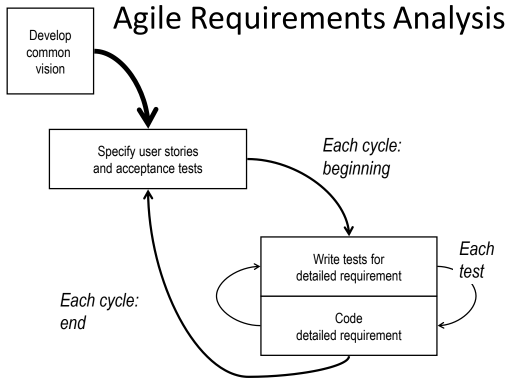
A Change In a Requirement Causes Changes In Other Artifacts

Artifact	Original version	Revised version
Requirement	The title of a DVD shall consist of between 1 and 15 English characters.	The title of a DVD shall consist of between 1 and 15 characters, available in English, French and Russian.
Design element		

Code	class DVD
	{ String title ….
	}
		class DVD
		{ Title title ….
		}
		class Title…

Inspection report	Inspection # 672: 4 defects; Follow-up inspection #684.	Inspection # 935: 1 defect; no follow-up inspection required.
Test report	Test # 8920 …	Test # 15084 …

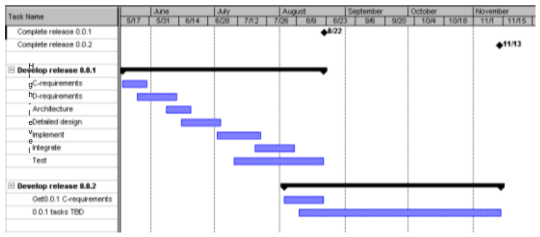
Agile Requirements Analysis


Develop common vision
Specify user stories and acceptance tests
Each cycle: beginning
Write tests for detailed requirement
Each test
Code detailed requirement
Each cycle: end

Updating Project Plan After Obtaining High-Level Requirements

	Status after initial draft	*Status after obtaining high-level requirements*
Milestones	Initial	**More milestones; more specific**
Risks	Identify initial risks	**Retire risks identified previously; identify more risks now that more is known about the project**
Schedule	Very rough	**Preliminary project schedule**
Personnel	Designate high-level requirements engineers	**Designated engineers for detailed requirements analysis**
Cost Estimation	Very rough	**First estimates based on job content**

Typical Schedule Following High-Level Requirements Analysis



19