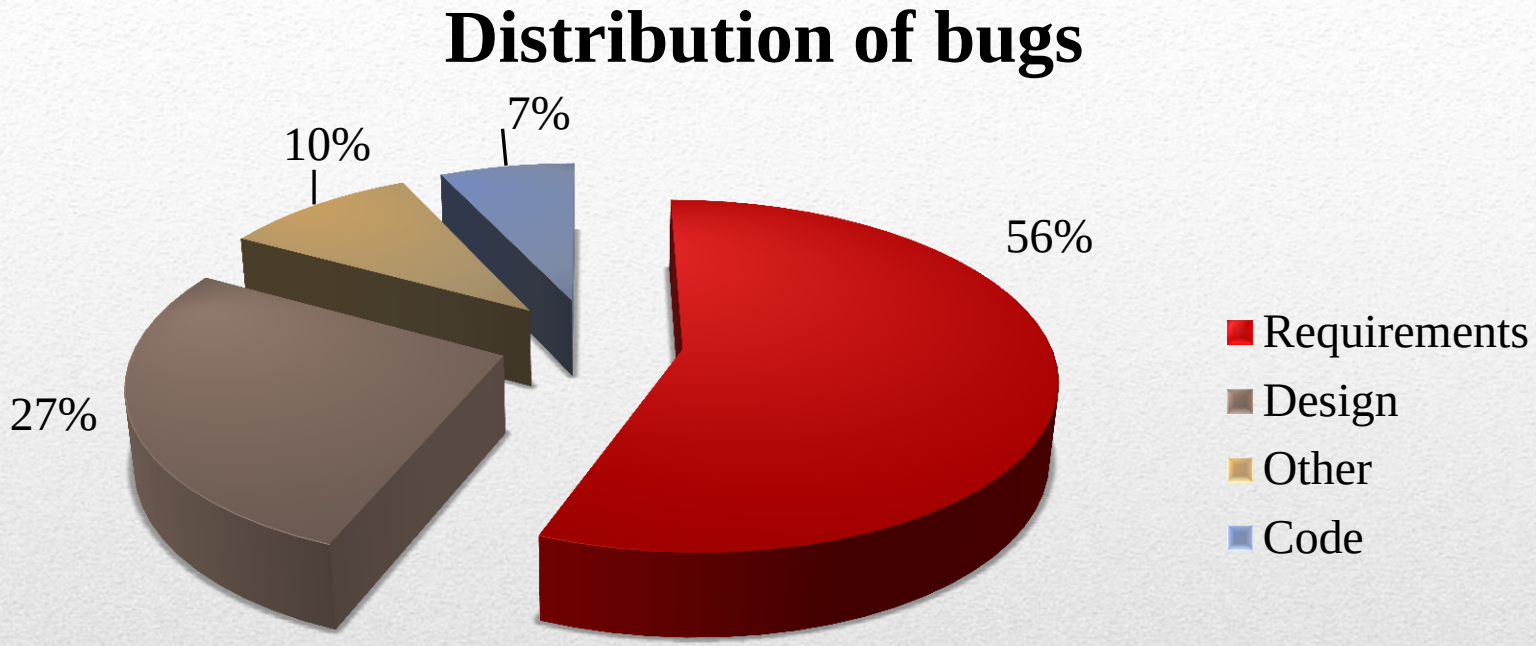
# Requirements Based Testing

PhD. Maryna Didkovska

# Requirement. Definition

- Requirements are a specification of what should be implemented.
- Requirement describes the need without describing any implementation details of a solution to that need
- *what*, but *not how* paradigm


Distribution of bugs
7%
10%
56%
27%
Requirements
Design
Other
Code

# Importance

| Phase in Which Found | Cost Ratio |
|---|---|
| Requirements | 1 |
| Design | 3-6 |
| Coding | 10 |
| Unit/Integration Testing | 15-40 |
| System/Acceptance Testing | 30-70 |
| Production | 40-1000 |

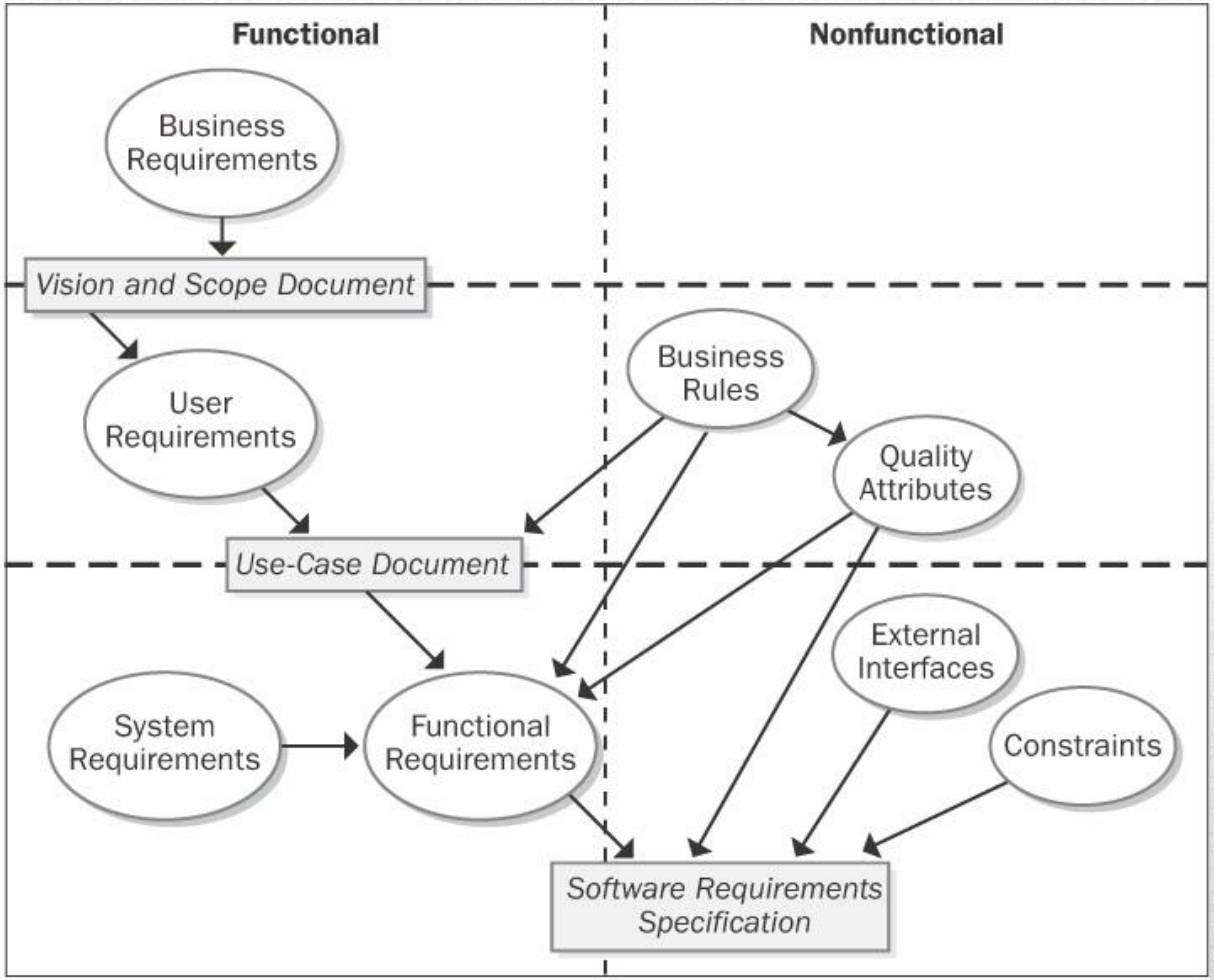
# Relative cost to fix an error

- Requirements and specifications are incomplete.
- Requirements and specifications change too often.
- There is a lack of user input (to requirements).

# Why software projects fail?

1. What the stakeholders expect will be delivered?
2. What the developers are expected to deliver?
3. What the testers are expected to test?

# Main questions




# Levels and Types

| Important Primarily to Users | Important Primarily to Developers |
|---|---|
| Availability | Maintainability |
| Efficiency | Portability |
| Flexibility | Reusability |
| Integrity | Testability |
| Interoperability | |
| Reliability | |
| Robustness | |
| Usability | |

# Quality attributes

- Unambiguous
- Complete
- Correct
- Feasible
- Necessary
- Prioritized
- Verifiable

# Requirement Statement Characteristics

- Complete
- Consistent
- Modifiable
- Traceable

# Requirements Specification Characteristics

| Relative Weights | 2 | 1 | | | 1 | | 0.5 | | |
|---|---|---|---|---|---|---|---|---|---|
| Feature | Relative Benefit | Relative Penalty | Total Value | Value % | Relative Cost | Cost % | Relative Risk | Risk % | Priority |
| *Some feature* … | 2 | 4 | 8 | 5.2 | 1 | 2.7 | 1 | 3.0 | 1.22 |
| Totals | 53 | 49 | 155 | 100.0 | 37 | 100.0 | 33 | 100.0 | |

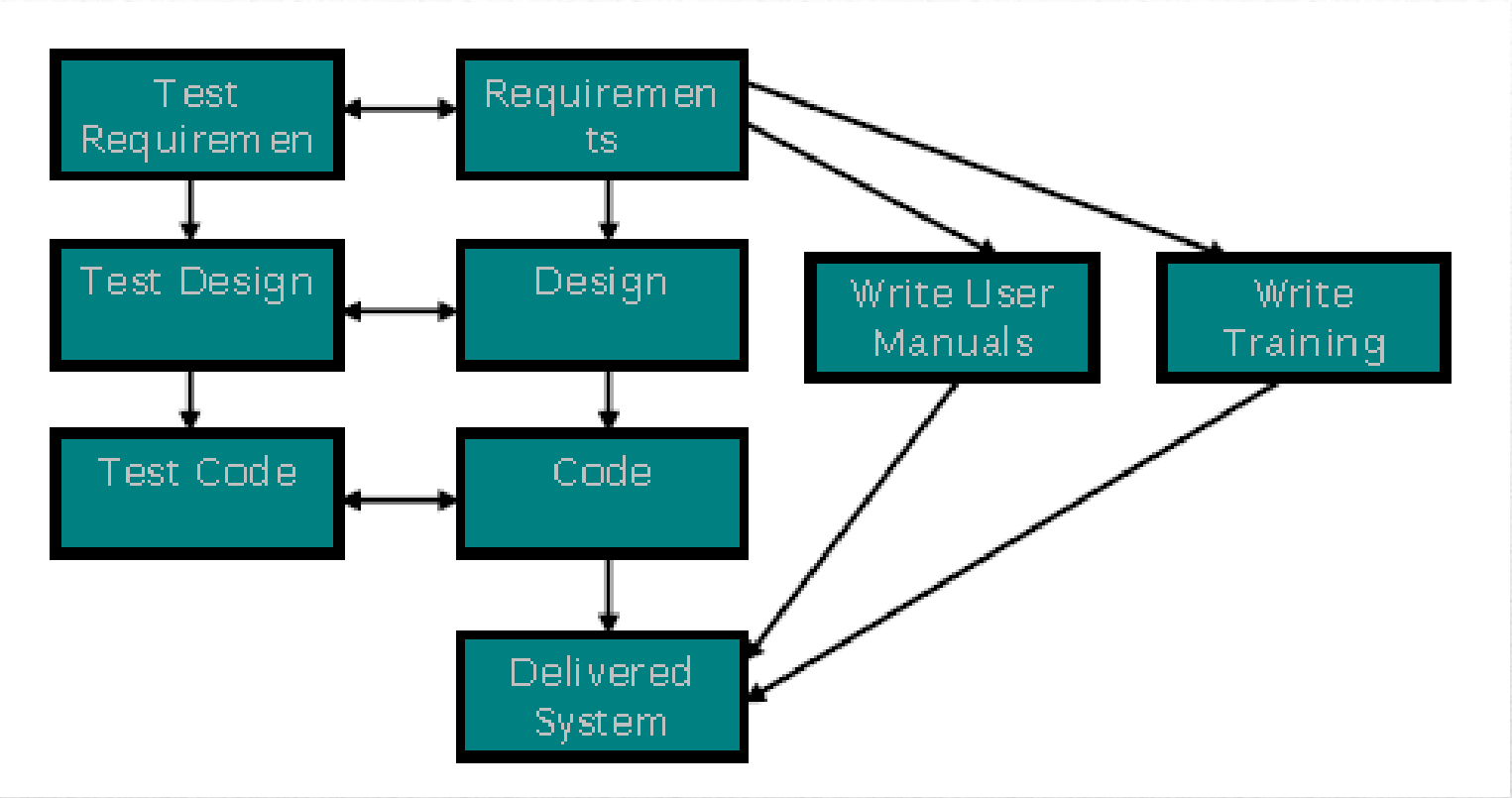
$$\text{priority} = \frac{\text{value \%}}{(\text{cost \%} * \text{cost weight}) + (\text{risk \%} * \text{risk weight})}$$

# Setting Requirement Priorities

| User Requirement | Functional Requirement | Design Element | Code Module | Test Case |
|---|---|---|---|---|
| UC-28 | catalog.query.sort | Class catalog | catalog.sort() | search.7<br>search.8 |
| UC-29 | catalog.query.<br>import | Class catalog | catalog.<br>import()<br>catalog.<br>validate() | search.12<br>search.13<br>search.14 |

Changes are not so dangerous
when you can manage them

# Requirements Traceability




# Requirements' impact

1. Validate requirements against objectives
2. Apply scenarios against requirements
3. Perform initial ambiguity review
4. Perform domain expert reviews
5. Create cause-effect graph
6. Logical consistency check by RBT Tool
7. Review of test cases by specification writers
8. Review of test cases by users
9. Review of test cases by developers
10. Walk test cases through design
11. Walk test cases through code
12. Execute test cases against code

# RBT Process

- 1. Review requirements documents
- 2. Test the requirements
- 3. Define acceptance criteria

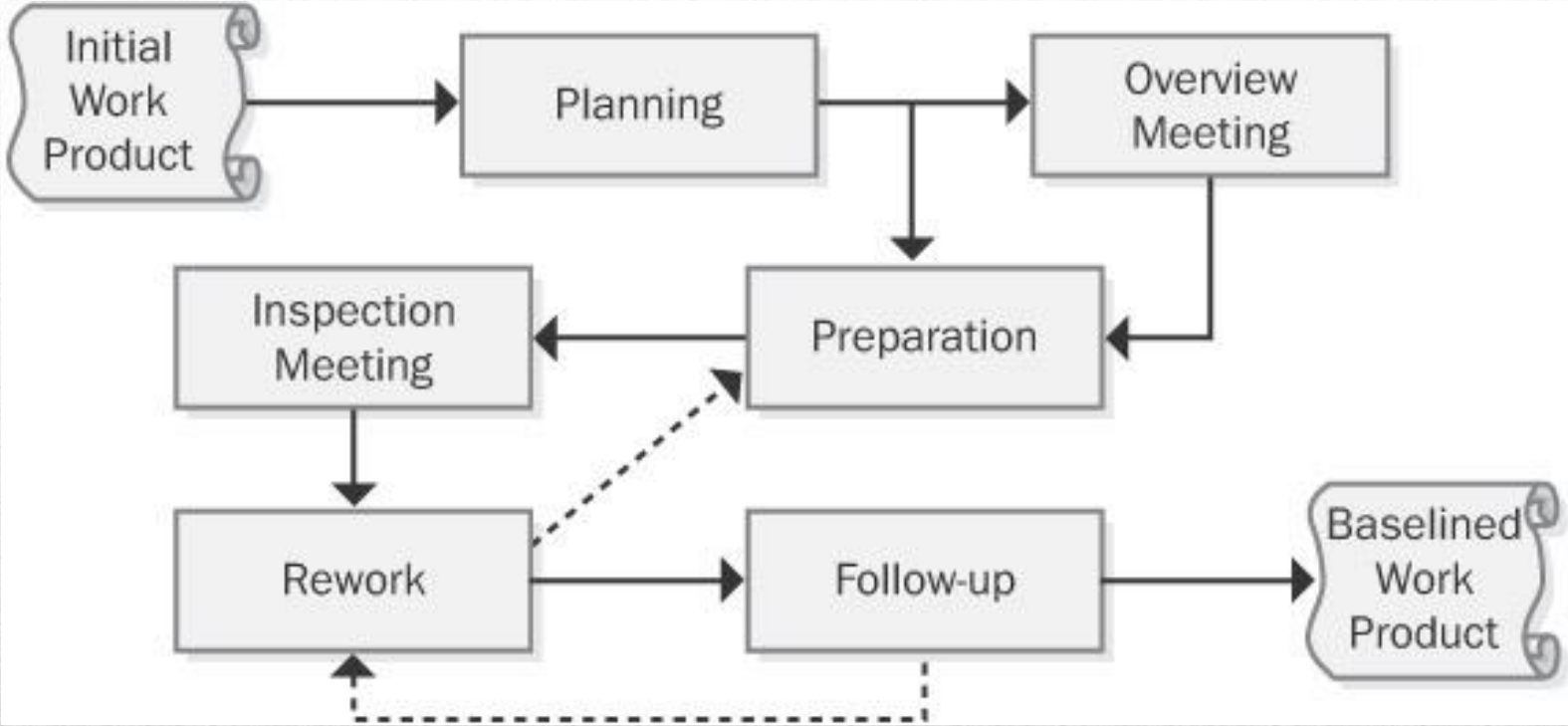
# Validating the requirements

- The **author of the work** product and perhaps peers of the author
- The **author of any predecessor work** product or specification for the item being inspected
- **People who will do work** based on the item being inspected
- People who **are responsible for work products that interface with the item** being inspected

# Review: Participants

- Author
- Moderator
- Reader
- Recorder

# Review: Roles




# Review: Stages

- Lets you define the data items included in a change request
- Lets you define a state-transition model for the change-request life cycle
- Enforces the state-transition model so that only authorized users can make the permitted status changes
- Records the date of every status change and the identity of the person who made it
- Lets you define who receives automatic e-mail notification when an originator submits a new request or when a request's status is updated

# Change-Control Tools

1. **Context free questions**. The traditional questions are:

*Who will use this feature? What does this user want to do with it? What do they lose? What else does this user want to do in conjunction with this feature? Who is not allowed to use this product or feature and what security is in place to prevent them?*

2. **Writing test cases**

*begin writing check lists and test cases early in requirements development.*

3. **Models, Diagrams**

*Requirements document is often long and hard to understand.*

*Requirements described on page 1 can contradict the requirement on page 100. It's very hard to remember everything*.

# Testing Requirements: Basic Techniques of Analyzing

- is a test of the requirements to insure that they are written in a clear, concise and unambiguous manner.
- occurs prior to the review of requirements for content by the domain experts.
- is intended  to provide the domain experts with a better quality set of requirements to work from, in order to identify missing requirements, and improve the content of all requirements.

# Ambiguity Review

| Ambiguous Terms | Ways to Improve Them |
|---|---|
| **acceptable, adequate** | Define what constitutes acceptability and how the system can judge this. |
| **as much as practicable** | Don't leave it up to the developers to determine what's practicable. Make it a TBD and set a date to find out. |
| **at least, at a minimum, not more than, not to exceed** | Specify the minimum and maximum acceptable values. |
| **between** | Define whether the end points are included in the range. |
| **depends on** | Describe the nature of the dependency. |
| **efficient** | Define how efficiently the system uses resources, how quickly it performs specific operations, or how easy it is for people to use. |
| **fast, rapid** | Specify the minimum acceptable speed at which the system performs some action. |
| **flexible** | Describe the ways in which the system must change in response to changing conditions or business needs. |

| Ambiguous Terms | Ways to Improve Them |
| --- | --- |
| **improved, better, faster, superior** | Quantify how much better or faster constitutes adequate improvement in a specific functional area. |
| **including, including but not limited to, and so on, etc., such as** | The list of items should include all possibilities. Otherwise, it can't be used for design or testing. |
| **maximize, minimize, optimize** | State the maximum and minimum acceptable values of some parameter. |
| **normally, ideally** | Also describe the system's behavior under abnormal or non-ideal conditions. |
| **optionally** | Clarify whether this means a system choice, a user choice, or a developer choice. |
| **reasonable, when necessary, where appropriate** | Explain how to make this judgment. |

| Ambiguous Terms | Ways to Improve Them |
|---|---|
| **robust** | Define how the system is to handle exceptions and respond to unexpected operating conditions. |
| **seamless, transparent, graceful** | Translate the user's expectations into specific observable product characteristics. |
| **several** | State how many, or provide the minimum and maximum bounds of a range. |
| **shouldn't** | Try to state requirements as positives, describing what the system will do. |
| **state-of-the-art** | Define what this means. |
| **sufficient** | Specify how much of something constitutes sufficiency. |
| **support, enable** | Define exactly what functions the system will perform that constitute supporting some capability. |
| **user-friendly, simple, easy** | Describe system characteristics that will achieve the customer's usage needs and usability expectations. |

- Ask users to describe how they will determine whether the product meets their needs and is fit for use.
- Base acceptance tests on usage scenarios

# Define acceptance criteria

- Schedule and cost overruns
- A product that doesn't satisfy user needs or doesn't meet user expectations
- A product that requires corrections and patches immediately following release
- Team member frustration, loss of morale, demotivation, and staff turnover
- Extensive rework
- Duplication of effort
- Missed market window or delayed business benefit
- Loss of market share or revenue
- Product returns or market rejection of the product
- Reduced feature set delivered
- Software that isn't testable

# Common Symptoms of Requirements Problems

- Fewer requirements defects
- Reduced development rework
- Fewer unnecessary features
- Lower enhancement costs
- Faster development
- Fewer miscommunications
- Reduced scope creep
- Reduced project chaos
- More accurate system-testing estimates
- Higher customer and team member satisfaction

# Benefits of good requirements