2. REQUIREMENTS DEVELOPMENT

CH05. **FINALIZING THE REQUIREMENTS DEVELOPMENT**

SOFTWARE REQUIREMENT ENGINEERING (UNDERGRADUATE)

PURPOSES OF PROTOTYPES

- Clarify, complete, and validate requirements
- Explore design alternatives
- Create a subset that will grow into the ultimate product

PAPER PROTOTYPES

- A paper prototype is a cheap, fast, and low-tech way to explore how a portion of an implemented system might look
  - helps to test whether users and developers hold a shared understanding of the requirements
  - involve tools no more sophisticated than paper, index cards, sticky notes, and whiteboards
  - facilitates rapid iteration, and iteration is a key success factor in requirements development
  - is an excellent technique for refining the requirements prior to designing detailed user interfaces, constructing an evolutionary prototype, or undertaking traditional design and construction activities
  - It also helps the development team manage customer expectations

MOCK-UPS

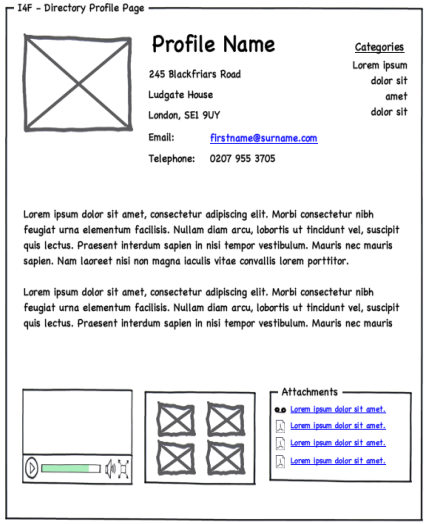
- A mock-up is also called a *horizontal prototype* (User interface Design)
  - Such a prototype focuses on a portion of the user interface; it doesn't deisgn into all the architectural layers or into detailed functionality
  - This type of prototype lets you explore some specific behaviours of the intended system, with the goal of refining the requirements

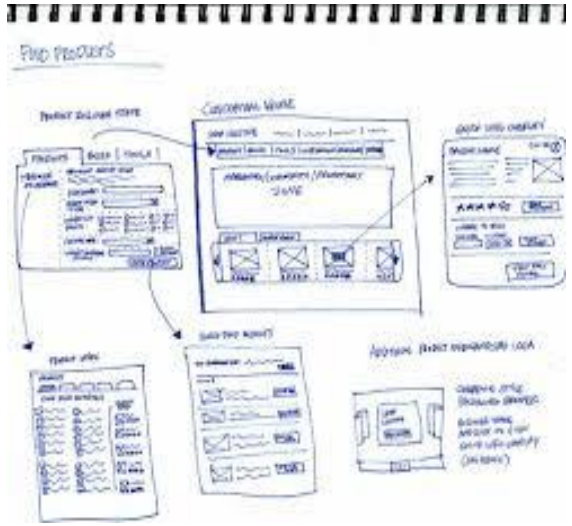
MOCK-UPS

- The mock-up helps users judge whether a system based on the prototype will let them do their job in a reasonable way
- A mock-up implies behaviour without actually implementing it
- Mock-ups can demonstrate the functional options the user will have available, the look and feel of the user interface (colours, layout, graphics, controls), and the navigation structure
- When working with a throwaway mock-up prototype, the user should focus on broad requirements and workflow issues without becoming distracted by the precise appearance of screen elements

WIREFRAME

- A wireframe is a particular approach to throwaway prototyping commonly used for custom user interface design and website design
- A better understanding of three aspects of a website:
  - The conceptual requirements (are used to understand a subject matter, is useful for working with users to understand the types of activities they might want to perform at the screen)
  - The information architecture or navigation design
  - The high-resolution, detailed design of the pages





PROOFS OF CONCEPT

- A proof of concept is also known as a *vertical prototype* (architecture design)
  - Implements a slice of application functionality from the user interface through all the technical services layers
- A proof-of-concept prototype works like the real system is supposed to work because it touches on all levels of the system implementation

THROWAWAY VS. EVOLUTIONARY

TABLE 15-1 Typical applications of software prototypes

	Throwaway	Evolutionary
Mock-up	■ Clarify and refine user and functional requirements. ■ Identify missing functionality. ■ Explore user interface approaches.	■ Implement core user requirements. ■ Implement additional user requirements based on priority. ■ Implement and refine websites. ■ Adapt system to rapidly changing business needs.
Proof of concept	■ Demonstrate technical feasibility. ■ Evaluate performance. ■ Acquire knowledge to improve estimates for construction.	■ Implement and grow core multi-tier functionality and communication layers. ■ Implement and optimize core algorithms. ■ Test and tune performance.

CLASSES OF PROTOTYPE ATTRIBUTES

Scope

- A mock-up prototype (sample functionality) focuses on the user experience; a proof-of-concept prototype explores the technical soundness of a proposed approach (e.g. ATM)

Future use

- A throwaway prototype is discarded after it has been used to generate feedback, whereas an evolutionary prototype grows into the final product through a series of iterations

Form

- A paper prototype is a simple sketch drawn on paper, a whiteboard, or in a drawing tool. An electronic prototype consists of working software for just part of the solution

POSSIBLE WAYS OF INCORPORATING PROTOTYPES INTO SDLC

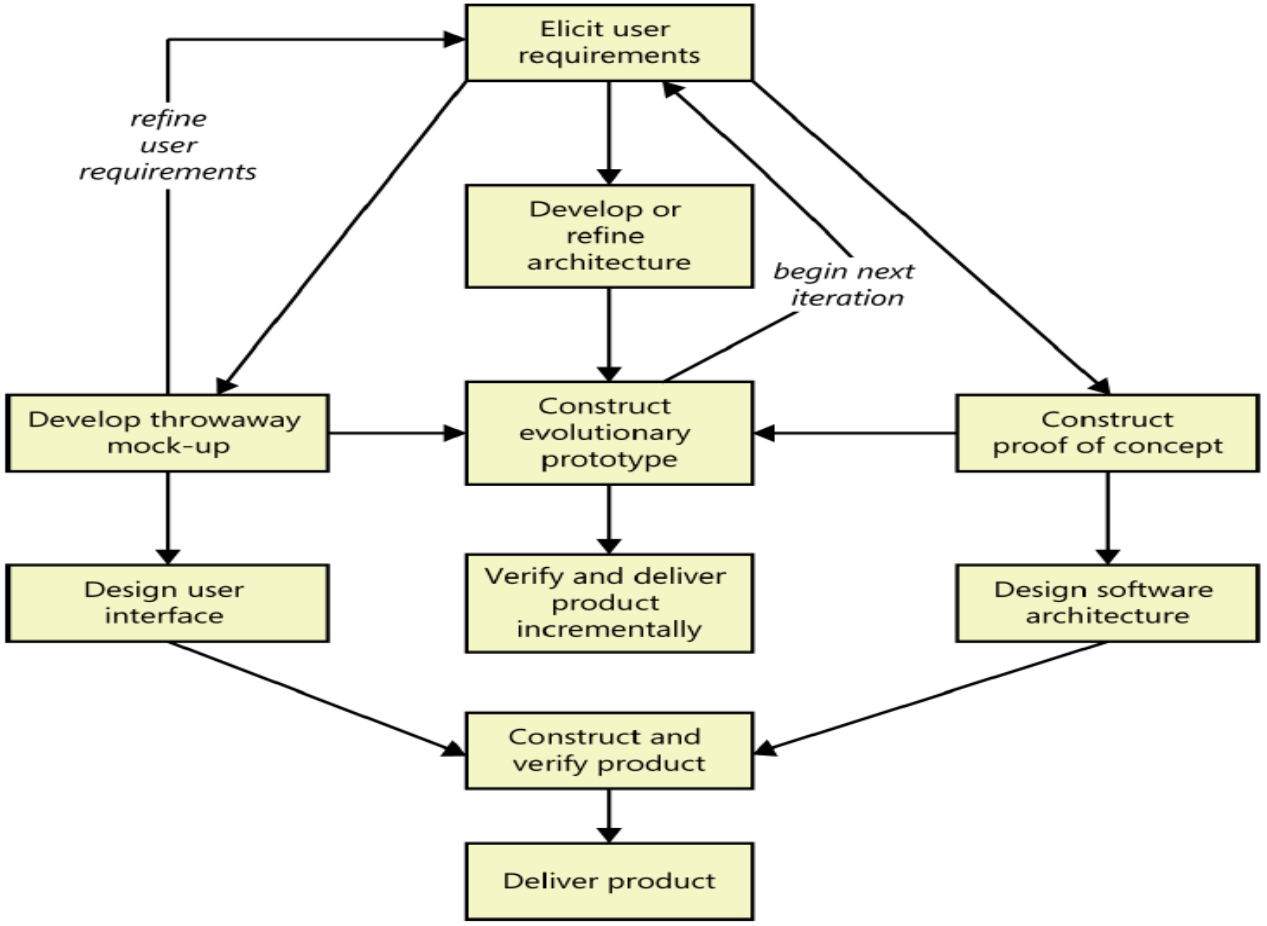


FIGURE 15-1 Several possible ways to incorporate prototyping into the software development process.

WORKING WITH PROTOTYPES

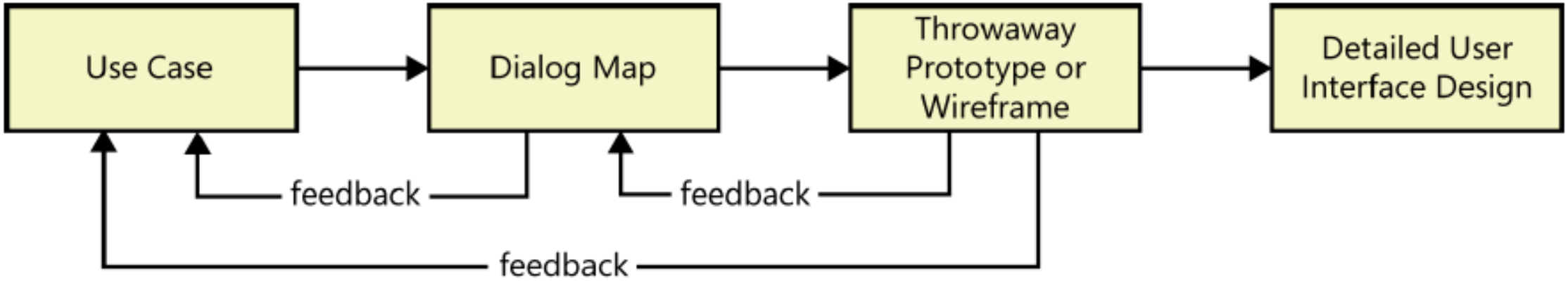


FIGURE 15-2 Activity sequence from use cases to user interface design using a throwaway prototype.

- Dialog Map is a user interface modelled in the form of a state-transition diagram representing different interaction and navigation map (page 235-237)

RISKS OF PROTOTYPING

- Pressure to release the prototype
- Distraction by details (user becomes obsessed, not cover all the options user wanted)
- Unrealistic performance expectations
- Investing excessive effort in prototypes

PROTOTYPING SUCCESS FACTORS

- Include prototyping tasks in your project plan. Schedule time and resources to develop, evaluate, and modify the prototypes
- State purpose of each prototype before you build it, and explain what will happen with outcome: either discard (or archive) the prototype, retaining the knowledge it provided, or build upon it to grow it into the ultimate solution
- Plan to develop multiple prototypes. You'll rarely get them right on the first try, which is the whole point of prototyping
- Create throwaway prototypes as quickly and cheaply as possible. Invest the minimum amount of effort that will answer questions or resolve requirements uncertainties
- Don't prototype requirements that you already understand, except to explore design alternatives
- Don't expect a prototype to replace written requirements

WHY PRIORITIZE REQUIREMENTS?

- When customer expectations are high and timelines are short, you need to make sure the product delivers the most critical or valuable functionality as early as possible
- Prioritization is a way to deal with competing demands for limited resources

PRIORITIZATION PRAGMATICS

- Successful prioritization requires an understanding of six issues:
  - The needs of the customers
  - The relative importance of requirements to the customers
  - The timing at which capabilities need to be delivered
  - Requirements that serve as predecessors for other requirements
  - Which requirements must be implemented as a group
  - The cost to satisfy each requirement (hard/soft satisfaction)

PRIORITIZATION TECHNIQUES

In or out

- The simplest of all prioritization methods is to have a group of stakeholders work down a list of requirements and make a binary decision: is it in, or is it out?

Pairwise comparison and rank ordering

- People sometimes try to assign a unique priority sequence number to each requirement
- Rank ordering a list of requirements involves making pairwise comparisons between all of them so you can judge which member of each pair has higher priority

PRIORITIZATION TECHNIQUES

- The four capitalized letters in the MoSCoW prioritization scheme stand for four possible priority classifications for the requirements in a set:
  - Must: the requirement must be satisfied for the solution to be considered a success
  - Should: the requirement is important and should be included in the solution if possible, but it's not mandatory to success
  - Could: it's a desirable capability, but one that could be deferred or eliminated. Implement it only if time and resources permit
  - Won't: this indicates a requirement that will not be implemented at this time but could be included in a future release

PRIORITIZATION TECHNIQUES

Three-level scale

- High-priority requirements are both important (customers need the capability) and urgent (customers need it in the next release)
- Medium-priority requirements are important (customers need the capability) but not urgent (they can wait for a later release)
- Low-priority requirements are neither important (customers can live without the capability if necessary) nor urgent (customers can wait, perhaps forever)

Importance of Requirements			
Urgency of Requirements	Low/Low	Medium/Low	High/Low
	Low/Medium	Medium/Medium	High/Medium
	Low/High	Medium/High	High/High

MULTIPASS PRIORITIZATION

- When performing a prioritization analysis with the three-level scale, you need to be aware of requirement dependencies
- You'll run into problems if a high-priority requirement is dependent on another that is ranked lower in priority and hence planned for implementation later on

PRIORITIZATION : RESOURCE ALLOCATION

- Prioritization is about thoughtfully allocating limited resources to achieve the maximum benefit from the investment an organization makes in a project.
- One way to make prioritization more tangible is to cast it in terms of an actual resource: **money**, in this case, it's just play with money.

EXAMPLE OF $100 IN PRIORITIZATION

- Give the prioritization team 100 imaginary dollars to work with.
- Team members allocate these dollars to "buy" items that they would like to have implemented from the complete set of candidate requirements.
- They weight the higher-priority requirements more heavily by allocating more dollars to them.
- If one requirement is three times as important to a stakeholder as another requirement, she would assign perhaps nine dollars to the first requirement and three dollars to the second.
- But 100 dollars is all the prioritizers get—when they are out of money, nothing else can be implemented, at least not in the release they are currently focusing on.
- One approach is to have different participants in the prioritization process perform their own dollar allocations, then add up the total number of dollars assigned to each requirement to see which ones collectively come out as having the highest priority.

PRIORITIZATION BASED ON VALUE, COST, AND RISK

- Typical participants in the prioritization process include:
  - The project manager or business analyst who leads the process, arbitrates conflicts, and adjusts prioritization data received from the other participants if necessary
  - Customer representatives such as product champions, a product manager, or a product owner, who supply the benefit and penalty ratings
  - Development representatives who provide the cost and risk ratings

STEPS TO USE PRIORITIZATION MODEL

1. List in the spreadsheet all the features, use cases, user stories, or functional requirements that you want to prioritize against each other.
2. Have the **customer representatives** estimate the relative benefit each feature would provide to the customer or to the business on a scale of 1 to 9. A rating of 1 indicates that no one would find it useful; 9 means that it would be **extremely valuable**.
3. Estimate the relative penalty that the customer or the business would suffer if each feature were not included. Again, use a scale of 1 to 9. A rating of 1 means that no one will be upset if it's absent; 9 indicates **a serious downside**. Requirements with both a **low benefit** and a **low penalty** add cost but little value.
4. The spreadsheet calculates the total value for each feature as the sum of its benefit and penalty scores.

STEPS TO USE PRIORITIZATION MODEL

5. Have **developers** estimate the relative **cost** of implementing each feature, again on a scale of 1 (quick and easy) to 9 (time-consuming and expensive). The spreadsheet will calculate the percentage of the total cost that each feature contributes.
6. Developers rate the relative technical (not business) **risk** associated with each feature on a scale of 1 to 9. Technical risk is the probability of not getting the feature right on the first try. A rating of 1 means you can program it in your sleep. A 9 indicates serious concerns about **feasibility**, the lack of necessary expertise on the team, the use of unfamiliar tools and technologies, or concern about the amount of complexity hidden within the requirement. The spreadsheet will calculate the percentage of the total risk that comes from each feature.

$$\text{Priority} = \frac{\text{value\%}}{\text{cost\%} + \text{Risk}}$$

STEPS TO USE PRIORITIZATION MODEL (CNTD.)

TABLE 16-1 Sample prioritization matrix for the Chemical Tracking System

Relative weights		**2**	**1**			**1**		**0.5**		
Feature		Relative benefit	Relative penalty	Total value	Value %	Relative cost	Cost %	Relative risk	Risk %	Priority
1.	Print a material safety data sheet.	2	4	8	5.2	1	2.7	1	3.0	0.91
2.	Query status of a vendor order.	5	3	13	8.4	2	5.4	1	3.0	1.0
3.	Generate a chemical stockroom inventory report.	9	7	25	16.1	5	13.5	3	9.1	0.68
4.	See history of a specific chemical container.	5	5	15	9.7	3	8.1	2	6.1	0.87
5.	Search vendor catalogs for a specific chemical.	9	8	26	16.8	3	8.1	8	24.2	0.83
6.	Maintain a list of hazardous chemicals.	3	9	15	9.7	3	8.1	4	12.1	0.68
7.	Change a pending chemical request.	4	3	11	7.1	3	8.1	2	6.1	0.64
8.	Generate a laboratory inventory report.	6	2	14	9.0	4	10.8	3	9.1	0.59
9.	Check training database for hazardous chemical training record.	3	4	10	6.5	4	10.8	2	6.1	0.47
10.	Import chemical structures from structure drawing tools.	7	4	18	11.6	9	24.3	7	21.2	0.33
	Totals	**53**	**49**	**155**	**100.0**	**37**	**100.0**	**33**	**100.0**	

VALIDATION AND VERIFICATION

- Verification determines whether the product of some development activity meets its requirements (doing the thing right).
- Validation assesses whether a product satisfies customer needs (doing the right thing). Requirements validation activities attempt to ensure that:
  - The software requirements accurately describe the intended system capabilities and properties that will satisfy the various **stakeholders' needs**.
  - The software requirements are correctly **derived from the business requirements**, system requirements, business rules, and other sources.
  - The requirements are complete, feasible, and verifiable.
  - All requirements are necessary, and the entire set is sufficient to meet the business objectives.
  - All requirements representations are consistent with each other (no conflicts).
  - The requirements provide an adequate basis to proceed with design and construction.

REVIEWING REQUIREMENTS

- Informal review approaches include:
  - Peer desk check, in which you ask one colleague to look over your work product.
  - Pass around, in which you invite several colleagues to examine a deliverable concurrently.
  - Walkthrough, during which the author describes a deliverable and solicits (asks) comments on it.
- Formal peer reviews follow a well-defined process. A formal requirements review produces a report that identifies the material examined, the reviewers, and the review team's judgment as to whether the requirements are acceptable.
  - The best-established type of formal peer review is called an **inspection**. Inspection of requirements documents is one of the highest-leverage software quality techniques available.

THE INSPECTION PROCESS

Participants

- The author of the work product and perhaps peers (similar competence) of the author (Analyst)
- People who are the sources of information that fed into the item being inspected (Customer)
- People who will do work based on the item being inspected (Developer)
- People who are responsible for interfacing systems that will be affected by the item being inspected

Inspection roles

- Author
- Moderator
- Reader
- Recorder

THE INSPECTION PROCESS

Entry criteria

- The document conforms to the standard template and doesn't have obvious spelling, grammatical, or formatting issues (manuscript's initial review).
- Line numbers or other unique identifiers are printed on the document to facilitate referring to specific locations.
- All open issues are marked as TBD (to be determined) or accessible in an issue-tracking tool.
- The moderator didn't find more than three major defects in a ten-minute examination of a representative sample of the document.

THE INSPECTION PROCESS

Inspection stages

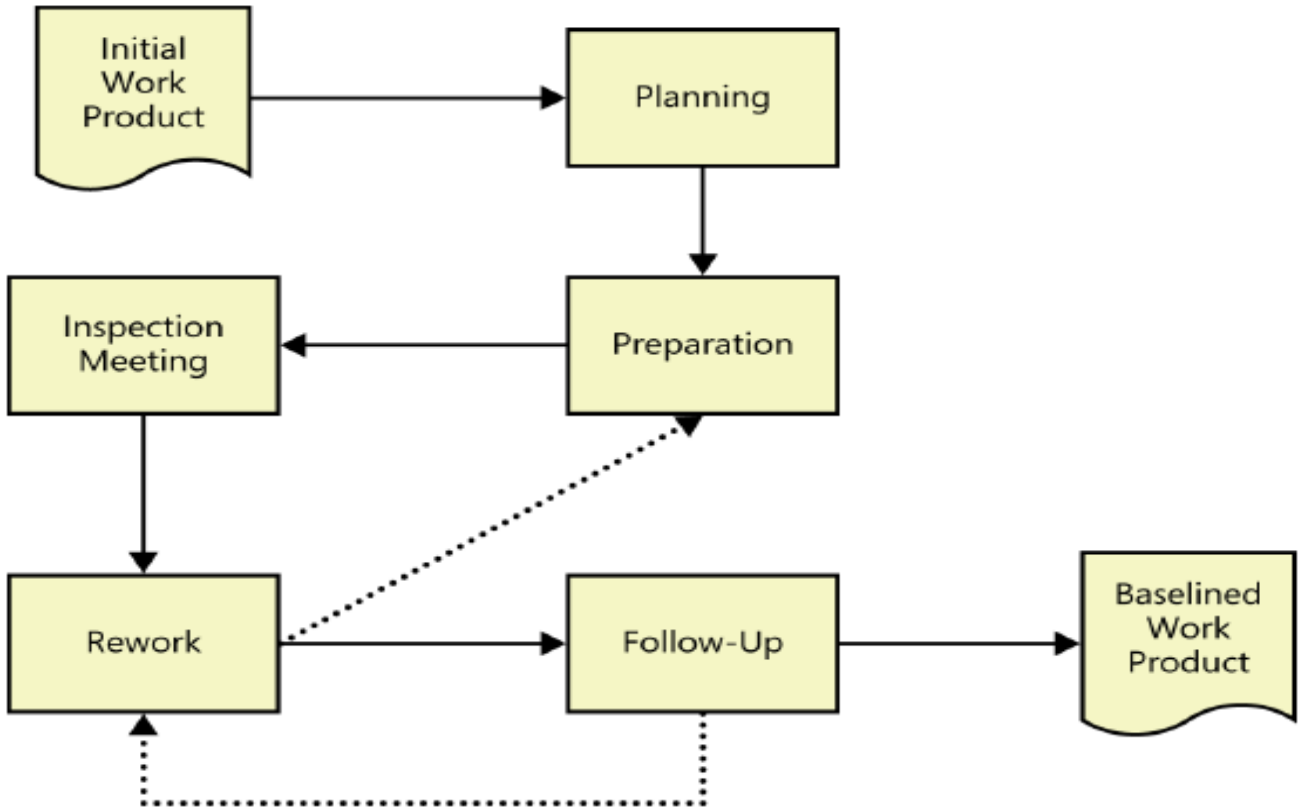


FIGURE 17-2 Inspection is a multistep process. The dotted lines indicate that portions of the inspection process might be repeated if reinspection is necessary because of extensive rework.

THE INSPECTION PROCESS

Exit criteria

- All issues raised during the inspection have been addressed.
- Any changes made in the requirements and related work products were made correctly.
- All open issues have been resolved, or each open issue's resolution process, target date, and owner have been documented.

DEFECT CHECKLIST

Completeness

- Do the requirements address all known customer or system needs?
- Is any needed information missing? If so, is it identified as TBD?
- Have algorithms inherently needed and specified to deliver the functional requirements?
- Are all external h/w, s/w, and communication interfaces defined?
- Is the expected behaviour documented for all anticipated error condition?
- Do the requirements provide an adequate basis for design and test?
- Is the implementation priority of each requirements included?
- Is each requirements in scope for the project release, or iteration?
- Is the requirements necessary for the product?

DEFECT CHECKLIST

Correctness

- Do any requirements conflict with or duplicate other requirements?
- Is each requirement written in clear, concise, unambiguous, and grammatically correct language?
- Are any specified error messages clear and meaningful?
- Are the requirements technically feasible and implementable within known constraints?

Quality Attributes

- Are all the quality attributes objectives properly specified and documented also quantified with the acceptable trade-offs specified?
- Are all the quality requirements measurable and verifiable?

REQUIREMENTS REVIEW TIPS

- Plan the examination – focus on specific section of document rather read the whole
- Start early – detecting major defect early
- Allocate sufficient time – specific hours in calendar time besides other tasks
- Provide context – give reviewers context for the document if they are not working on the same project
- Set review scope – tell reviewers what material to examine, where to focus their attention, and what issue to look for
- Limit re-reviews – don't ask anyone to review the same material more than three times
- Prioritize review areas – prioritize for review those portions of the requirements that are of risk or have functionality that will be used frequently

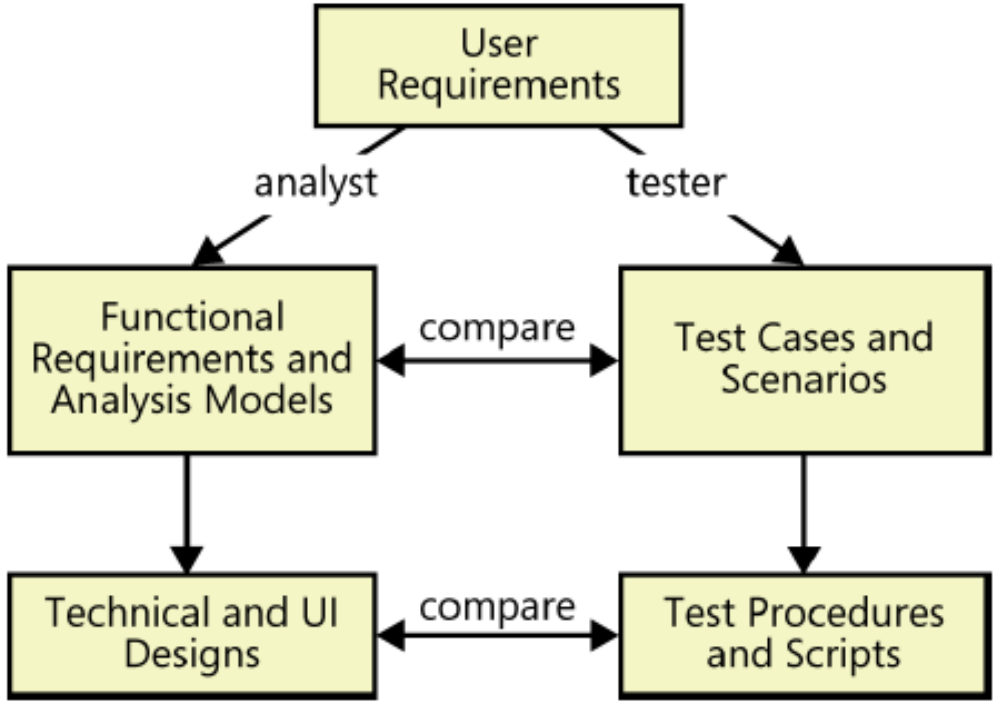
REQUIREMENTS REVIEW CHALLENGES

- Large requirements documents
- Large inspection teams
- Geographically separated reviewers
- Unprepared reviewers

PROTOTYPING IN REQUIREMENTS VALIDATION

- Prototypes are validation tools that make the requirements real
- All kinds of prototypes allow you to find missing requirements before more expensive activities like development and testing take place
- Prototypes also help confirm that stakeholders have a shared understanding of the requirements
- Proof-of-concept prototypes can demonstrate that the requirements are feasible

TESTING THE REQUIREMENTS



Project Name:

Test Case Template

Test Case ID: Fun_10	**Test Designed by:** <Name>
Test Priority (Low/Medium/High): Med	**Test Designed date:** <Date>
Module Name: Google login screen	**Test Executed by:** <Name>
Test Title: Verify login with valid username and password	**Test Execution date:** <Date>
Description: Test the Google login page	

Pre-conditions: User has valid username and password
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User: example@gmail.com	User should be able to login	User is navigated to	Pass	
2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database.

FIGURE 17-5 Development and testing work products are derived from a common source.

ACCEPTANCE CRITERIA

- Specific high-priority functionality that must be present and operating properly before the product could be accepted and used
- Essential non-functional criteria or quality metrics that must be satisfied
- Remaining major open issues and defects are resolved
- Specific legal, regulatory, or contractual conditions are fully satisfied
- Supporting transition, infrastructure, or other project requirements (e.g. training materials) are available

ACCEPTANCE TESTS

- Focus on testing the normal flows of the use cases and their corresponding exceptions, devoting less attention to the less frequently used alternative flows

WHY REUSE REQUIREMENTS?

- Faster delivery & Higher team productivity
- Fewer defects & Lower development costs
- Reduced rework & Save review time
- Accelerate the approval cycle
- Speed up other project activities, such as testing
- Improve your ability to estimate implementation effort if you have data available from implementing the same requirements on a previous project
- Improve functional consistency across related members of a product line or among a set of business applications (a set of products in a family is called software product line)
- Saves time for stakeholders, who then will not need to specify similar requirements repeatedly

DIMENSIONS OF REQUIREMENTS REUSE

Extend of Reuse

- Individual requirements statement
- A set of requirements and their associated design, code, and test elements

Extent of Modification

- Associated requirements attributes (priority, rationale, origin, etc.)
- Related information (test, design, constraints, data definition, etc.)

Reuse Mechanism

- Copy-and-paste from another specification
- Copy from a library of reusable requirements

REQUIREMENT EVOLUTION

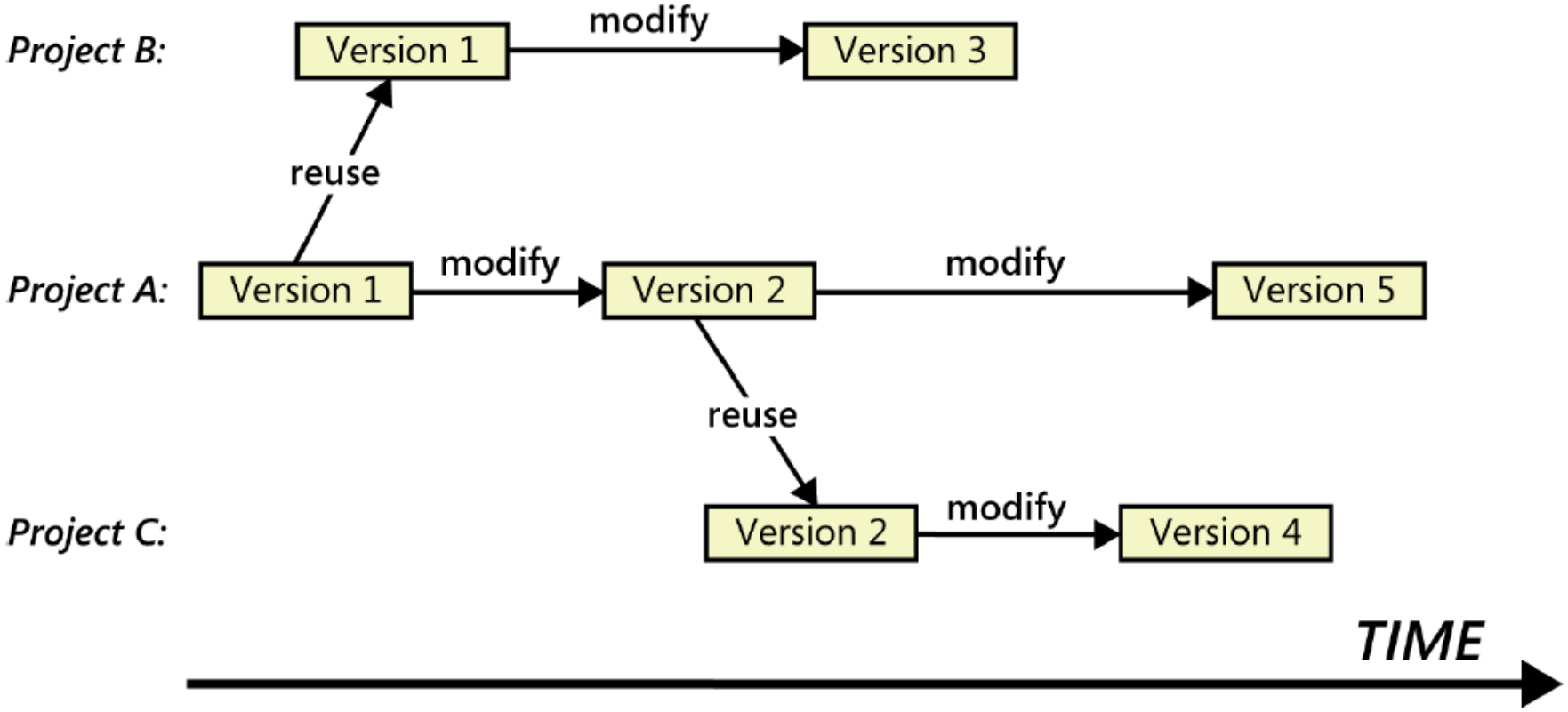


FIGURE 18-1 How a requirement can evolve through reuse in multiple projects.

REQUIREMENT PATTERNS

- Guidance of writing requirements (situation to which it is applicable)
- Content of the requirements (details modifications)
- Template of requirements (fill in the blanks)
- Example (illustrative requirements of this type)
- Consideration for development and testing (factors for developer/tester to keep in mind)

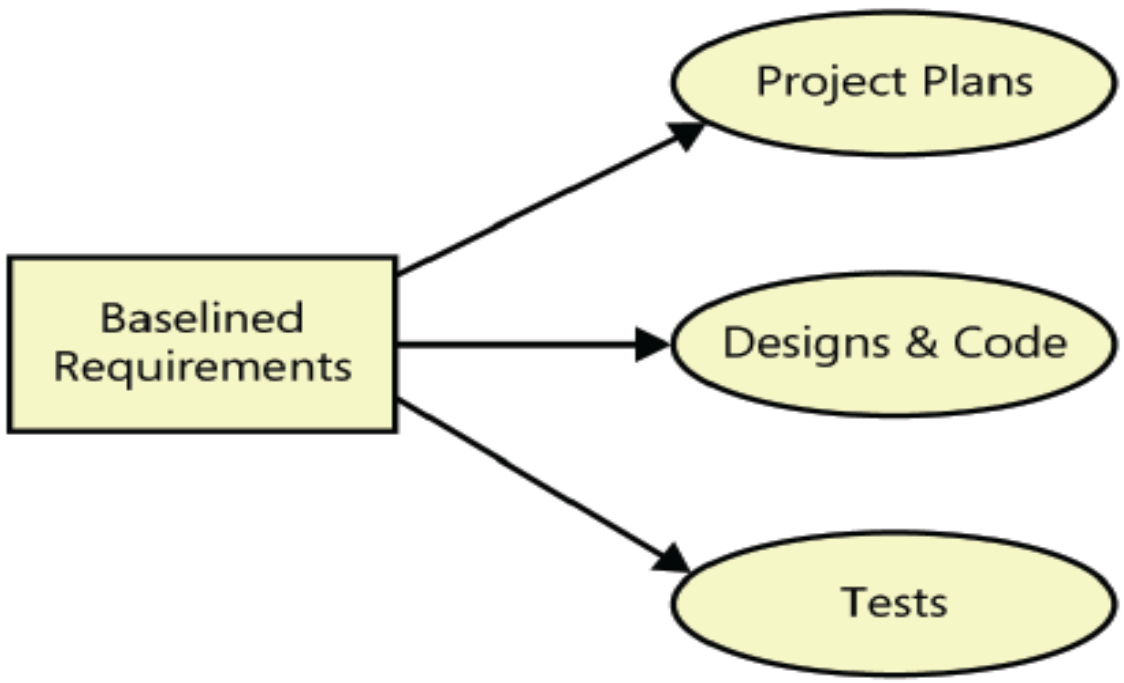
REUSE BARRIERS

- Missing or poor requirements
- NIH and NAH:
  - NIH means "not invented here" – less effort need to write new requirements than to understand and fix existing generic requirements from other organizations.
  - NAH, or "not applicable here"
- Writing style
- Inconsistent organization – organize the requirements in different ways: product feature, process
- Project type – requirements that are tightly coupled to specific implementation environment or platform
- Ownership – copyright of intellectual properties of requirements in other system development

REUSE SUCCESS FACTORS

- Repository
  - A single network folder that contains previous requirements documents
  - A collection of requirements stored in a requirements management tool that can be searched across projects
  - A database that stores sets of requirements selected from projects for their reuse potential and enhanced with keywords to help future BAs know their origin, judge their suitability, and learn about their limitations
- Interactions – use traceability links in a tool to identify the dependencies of reusable requirements
- Terminology – establish common terminology in reusing requirements
- Organizational culture – encourage requirements reuse

USE OF REQUIREMENTS



- Use requirements to size the project or iteration
- Base estimates on product size
- Update plans as requirements change
- Use requirement priorities to drive iterations

- Have developers review requirements
- Use quality attributes to drive architecture
- Allocate requirements to components
- Trace requirements to designs and code

- Start test design early
- Have users create acceptance tests
- Base system testing on requirements
- Trace requirements to tests

FIGURE 19-1 Requirements drive project planning, design, coding, and testing activities.

REFERENCES

- Wiegers, K., & Beatty, J. (2013). *Software requirements*. Pearson Education