
Sample

In Search of

Excellent

Requirements

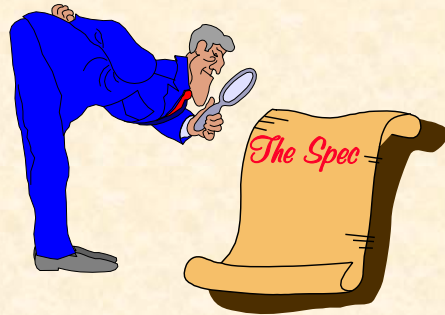


Karl E. Wieggers

PROCESS IMPACT 

www.processimpact.com

In Search of Excellent Requirements



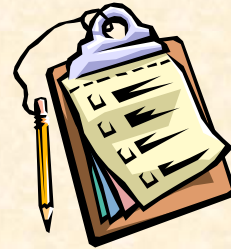
Karl E. Wiegiers
PROCESS IMPACT
www.processimpact.com

Notes:

- This is a comprehensive course on the processes and practices involved with eliciting, analyzing, specifying, validating, and managing the requirements for software and systems development projects.
- The practices apply to all types of projects, including end-user applications, web sites, embedded systems (real-time) products, and so on, when judiciously selected and adapted to suit the needs and reality of each project situation and the culture of each project team.

Course Agenda - 1

1. Introduction to Requirements Engineering
2. Requirements Development Process
3. Customer Involvement
4. Business Requirements
5. Requirements Elicitation
6. User Requirements
7. Business Rules
8. Requirements Specification
9. Quality Attributes



Notes:

- These are the modules covered in this course.

Module 10: Requirements Prioritization

Learning Objectives:

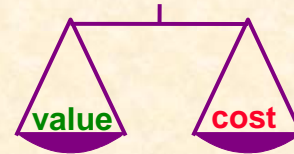
- ➔ Describe why requirements prioritization is important
- ➔ Use a three-level prioritization scale
- ➔ Use a spreadsheet to prioritize discretionary requirements

➤ Chapter 14 of *Software Requirements, 2nd Edition*

Notes:

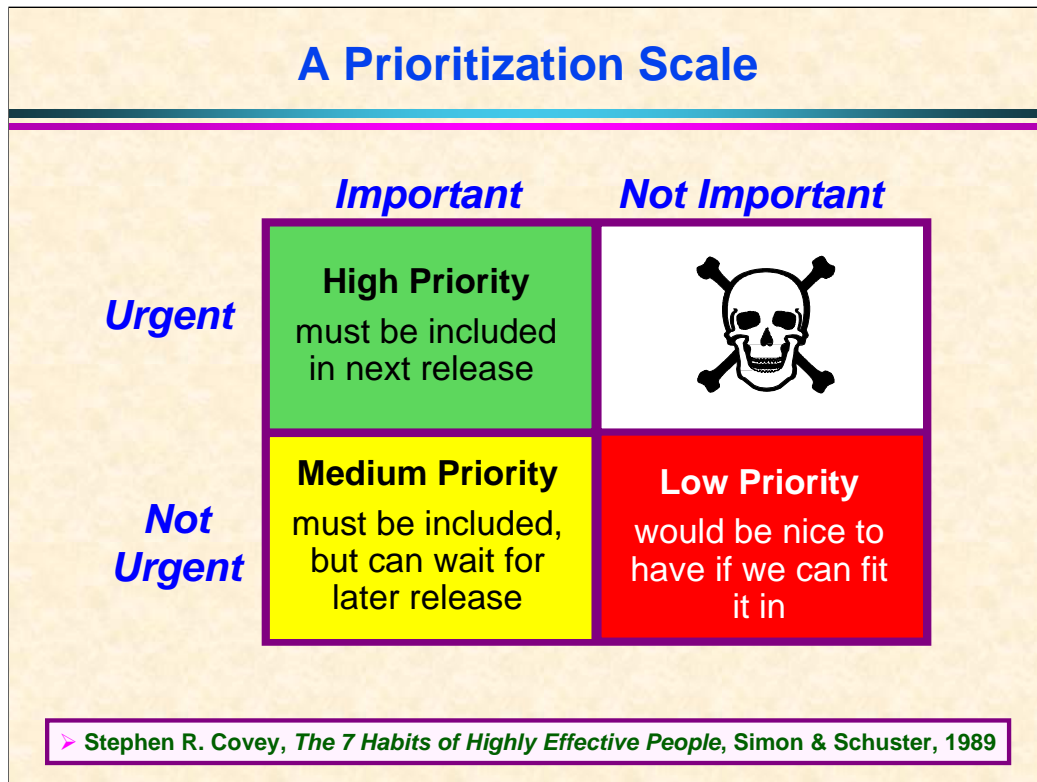
Requirements Prioritization

- Not everything can be top priority!
- Setting priorities will help you:
 - ✓ work on the right things first
 - ✓ make tradeoff decisions
 - ✓ deal with added and changed requirements
- Need to bypass politics and emotion.
 - ✓ favorable indicator: customer value (benefit + penalty)
 - ✓ unfavorable indicators: cost and technical risk
- Need to understand which requirements are most **important** and most **urgent**.



Notes:

- Prioritization becomes important during the “rapid descoping phase” at the end of so many software projects, when we have to decide what to defer in order to deliver something on schedule.
- If we can make those decisions in crisis mode at the end of the project, then we should be able to prioritize the critical product features, use cases, or functional requirements earlier in the project to make sure we can make sensible scheduling and trade-off decisions throughout the project.

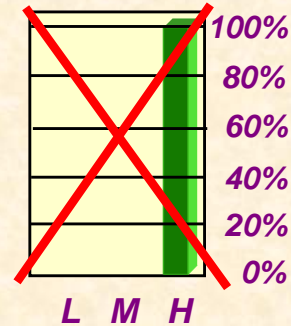


Notes:

- My prioritization scale combines the ideas of importance and urgency, from Stephen Covey's *The Seven Habits of Highly Effective People*:
- High priority items are important and urgent.
- Medium priority items are important but not so urgent.
- Low priority items are not so important and not urgent.
- The fourth quadrant contains requirements that seem to be urgent but really aren't important. Don't do those! They don't add sufficient value to the project.
- Prioritization scales all say essentially the same thing in different words. If there are three levels, they boil down to high, medium, and low.
- Everyone has to understand what the scale used means, so they classify the requirements consistently.

An Analytical Prioritization Procedure

1. Download requirements prioritization spreadsheet from www.processimpact.com.
2. Identify discretionary capabilities.
 - ✓ features, use cases, or functional reqs
3. Have product champions rate them:
 - ✓ relative benefit (1-9)
 - ✓ relative penalty (1-9)
4. Have developers rate them:
 - ✓ relative cost (1-9)
 - ✓ relative technical risk (1-9)
5. Spreadsheet calculates priority numbers.



Notes:

- If you can agree on priorities by negotiation, great! Always use the simplest prioritization approach you can.
- Sometimes, though, a more structured, analytical approach is needed.
- The next few slides describe a requirements prioritization spreadsheet that can be downloaded from the Process Impact web site. It is adapted from Quality Function Deployment.
- Here is the outline of the procedure that you use with this spreadsheet. The next few slides show an example.
- Only use this method on discretionary items, not top priority items. You're going to implement those in any case.
- You can do the prioritization on a group of use cases, features, or functional requirements; don't combine them, though. This example uses features.

In Search of Excellent Requirements

Karl E. Wieggers

Process Impact

Copyright © 2010 Karl E. Wieggers

“I’ll go find out what they want, and the rest of you start coding.” This caption from a cartoon is uncomfortably close to the way some software organizations still treat the requirements specification process. Contemporary definitions of “quality” include the concepts of both meeting stated specifications and satisfying the actual customer requirements, which sadly are not always the same thing on a software project.

Perhaps the greatest challenge facing the software developer is achieving a shared vision of the final product with the customer. All stakeholders in a project—developers, end users, software managers, customer managers—must reach a common understanding of what the product will be and do, or someone will be surprised when it is delivered. Surprises in software are almost never good news. Therefore, we need ways to accurately capture, interpret, and represent the voice of the customer when specifying the requirements for a software product [Wieggers, 2003].

As part of a multi-year process improvement effort [Wieggers, 1996], our small software group focused on improved requirements specifications as the starting point for increasing our software quality and productivity. Similar practices have been applied, with considerable success, in many other groups with which I have worked. Our quest for excellent requirements specifications involves six major techniques, which address the challenges of gathering, documenting, and validating user needs:

1. A high degree of customer participation in the development effort through a “product champion” model.
2. Gathering requirements from customer representatives by employing “use cases,” which focus on tasks the user must be able to accomplish with the product.
3. Preparation of structured specification documents in a format similar to the IEEE standard for software requirements specifications.
4. Construction of “dialog maps,” a variant of state transition diagrams that model complex user interfaces at a high level of abstraction.
5. Extensive use of prototypes to more fully elucidate user needs and explore alternative design solutions for satisfying them.
6. Constructing a requirements traceability matrix to link individual requirements to the design, code, and test elements ultimately constructed to implement and verify them.

While these practices have not been universally applied on every project, we invariably obtain better results when we use them than when we apply our previous, less structured requirements practices. Our experience indicates that these techniques can greatly reduce the expectation gap that often develops between the system the customer really needs to solve his problems, and the system the developer ultimately builds.

Characteristics of Excellent Requirements

High-quality software requirements have several well-defined characteristics, as shown in Table 1. As you write or review requirements specifications, keep these characteristics in mind. Any departures from them are likely to pose problems during the rest of the product development cycle, so correcting these problems early on can save considerable money, time, and aggravation.

Table 1. Characteristics of Excellent Requirements Specifications.

Complete	No requirements or necessary information should be missing. While missing requirements are difficult to identify, sometimes we know we are lacking certain information. Use TBD ("to be determined") as a flag to highlight this gap until the necessary information is obtained.
Consistent	Internal conflicts between requirements must be resolved before development can proceed. You don't know which one (if any) is correct until some further research is done.
Correct	Each requirement statement must accurately state a customer need. Only customer representatives can determine this, which is why it is essential to include customers or their surrogates in reviews of requirements documents.
Feasible	It must be possible to implement each stated requirement within the known constraints and limitations of the system and its environment.
Modifiable	To properly manage requirements, we must be able to maintain a history of changes made in each requirement. This requires that each requirement be uniquely labeled so that it can be referred to unambiguously.
Necessary	Each requirement should document something the customers really need. Avoid the temptation to gold-plate the requirements by adding features the developers are "just sure the users will love."
Prioritized	Assign an implementation priority to each requirement. If the requirements are all equally important, then we lose a degree of negotiation freedom if we have to respond to new requirements added during development, budget cuts, schedule overruns, or loss of project personnel. Consider a three-level priority scheme: High: must be present in the next release Medium: can be deferred to a subsequent release Low: would be nice to have, but we can live without it
Traceable	It is valuable to be able to link each software requirement to its source (a higher-level system requirement, a use case, or a voice of the customer statement). We also want to be able to link each software requirement to the design elements, source code elements, and test cases that are constructed to implement and verify the requirement.
Unambiguous	Ambiguity in requirements is a serious problem. The only way to ferret out ambiguity is to have a group of people representing different perspectives inspect the requirements as a team. Simply passing around the requirements document for comments is unlikely to reveal ambiguity. If a requirements statement is interpreted in different ways by different reviewers, but it makes sense to each of those reviewers, the ambiguity will never surface. (Actually, it will eventually surface, but late in the project, when it can cost a great deal to correct.)
Verifiable	Examine each requirement from the perspective of whether you can devise a small number of tests, or use other verification approaches such as inspection or demonstration, to determine whether the requirement has been properly implemented. In fact, a good guideline for the appropriate level of detail and granularity to write requirements is to make them individually testable.

Worksheets for “In Search of Excellent Requirements”

Checklist for Reviewing Software Requirements Specifications

Organization and Completeness

- Are all internal cross-references to other requirements correct?
- Are all requirements written at a consistent and appropriate level of detail?
- Do the requirements provide an adequate basis for design?
- Is the implementation priority of each requirement included?
- Are all hardware, external software, and communication interfaces defined?
- Have algorithms intrinsic to the functional requirements been defined?
- Does the SRS include all of the known customer or system needs?
- Is any necessary information missing from a requirement? If so, is it identified as TBD?
- Is the expected behavior documented for all anticipated error conditions?

Correctness

- Do any requirements conflict with or duplicate other requirements?
- Is each requirement written in clear, concise, unambiguous language?
- Is each requirement verifiable by testing, demonstration, review, or analysis?
- Is each requirement in scope for the project?
- Is each requirement free from content and grammatical errors?
- Can all of the requirements be implemented within known constraints?
- Are any specified error messages unique and meaningful?

Quality Attributes

- Are all performance objectives properly specified?
- Are all security and safety considerations properly specified?
- Are other quality attribute goals explicitly documented and quantified, with the acceptable tradeoffs specified?

Traceability

- Is each requirement uniquely and correctly identified?
- Is each software requirement traceable to a higher-level requirement (e.g., system requirement, use case)?

Special Issues

- Are all requirements actually requirements, not design or implementation solutions?
- Are the time-critical functions identified, and timing criteria specified for them?
- Have internationalization issues been adequately addressed?

