The Effect of software requirements analysis on project success and product quality

Vera Berenbaum

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

The Effect of Software Requirements Analysis on Project Success and Product Quality

by

Vera Berenbaum

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

August 2001
Rochester Institute of Technology
Department of Information Technology

Master of Science in Information Technology
Thesis Approval Form

Student Name: Vera Berenbaum

Project Title: The Effect of Software Requirements Analysis on Project Success and Product Quality

Project Committee

Name: Timothy D. Wells
Signature: ____________________________
Date: 6/27/01
Committee Chair

Name: Anne R. Haake
Signature: ____________________________
Date: 6/27/01
Committee Member

Name: Raymond D. Niemi
Signature: ____________________________
Date: 6/27/01
Committee Member
Release Permission Form

Department of Information Technology
Rochester Institute of Technology
B. Thomas Golisano College of Computing and Information Sciences

The Effect of Requirements Analysis on Project Success and Product Quality

I, Vera Berenbaum, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only, with my permission.

Vera Berenbaum

8/14/2001
Date
© Copyright 2001 Vera Berenbaum

All Rights Reserved
# Table of Contents

1. **ABSTRACT** ............................................................................................................. 1

2. **INTRODUCTION** ................................................................................................. 1

3. **METHODOLOGY** ................................................................................................. 3

4. **REQUIREMENTS ANALYSIS** ............................................................................... 4
   4.1 Requirements Analysis Defined........................................................................... 4
   4.2 Best Practices in Requirements Analysis ............................................................ 6
      4.2.1 General Overview ......................................................................................... 6
      4.2.2 Customer Involvement ................................................................................. 10
      4.2.3 User Involvement ......................................................................................... 12
      4.2.4 Requirements Validation Techniques ............................................................ 13
   4.3 Usefulness and Usability Analysis ....................................................................... 15
      4.3.1 Usefulness ..................................................................................................... 15
      4.3.2 Usability ........................................................................................................ 16
   4.4 Requirements Prioritization and Scope Definition ............................................... 22
   4.5 Requirement Change Control Management ....................................................... 27
      4.5.1 Change .......................................................................................................... 27
      4.5.2 Control .......................................................................................................... 31
   4.6 Best Practices in Requirements Change Control ................................................. 33
   4.7 Project-Related Reasons to Update Requirements after the Baseline ................. 35
   4.8 Organizational Benefits of Updating Requirements and Documenting System Functionality .................................................................................................................................................................................. 35
   4.9 Alternatives to the Waterfall Model .................................................................... 37
      4.9.1 Extreme Programming, Iterative Development and Requirements Analysis .... 37
      4.9.2 Rapid Development and Requirements Analysis ............................................. 41
   4.10 How Prevalent is Sound Requirements Analysis in the Software Development Industry? .................................................................................................................................................................................. 42

5. **IMPACT OF REQUIREMENTS ANALYSIS ON PRODUCT QUALITY** ............... 46
   5.1 Quality Management Depends on Sound Requirements ..................................... 46
      5.1.1 Quality Planning ............................................................................................. 48
      5.1.2 Quality Assurance ......................................................................................... 49
   5.2 Quality Acceptance Criteria ................................................................................ 49
   5.3 Product Success .................................................................................................... 50
      5.3.1 Definition of a Successful Product ................................................................ 50
      5.3.2 Usability ........................................................................................................ 51
      5.3.3 Short-term sales ............................................................................................. 51
      5.3.4 Long-term sales ............................................................................................. 51
      5.3.5 Subscription renewals ................................................................................... 51
      5.3.6 Customer feedback ....................................................................................... 52
   5.4 Ways In Which a Product Can Be Unsuccessful .................................................... 52
   5.5 System Acceptability ............................................................................................ 52
   5.6 Creative Ways to Re-Define Product Success ...................................................... 53
   5.7 Definition of Product Management .................................................................... 54
5.8 Effect of Requirements Analysis on Project Success ................................................. 54
5.9 Recommendations to the Product Manager ................................................................. 56
5.10 Assessing the Value of Sound Requirements Analysis .............................................. 57

6. IMPACT OF REQUIREMENTS ON PROJECT SUCCESS ............................................. 58
6.1 Emerging Trends in Project Management are Not Successful ...................................... 58
6.2 Why Shorten Projects? ............................................................................................... 61
6.3 How to Shorten Projects with Minimal Risk ............................................................... 62
6.4 The Definition of a Successful Software Project .......................................................... 65
6.5 Creative Ways to Redefine Project Success ................................................................. 66
6.6 How Successful Are Software Projects? ...................................................................... 67
6.7 Effect of Requirements Analysis on Project Success ................................................... 68
6.8 Recommendations to the Project Manager ................................................................. 68
   6.8.1 Do not limit the requirements analysis ................................................................. 68
   6.8.2 Do not undertake a project without understanding its full scope and the
        organization’s capabilities ..................................................................................... 69
   6.8.3 Resist shortening projects at the expense of quality and usability .......................... 69
   6.8.4 More Advice ...................................................................................................... 70

7. CASE STUDIES .............................................................................................................. 70
7.1 Case 1 – Requirements Analyst’s Account of Successful Project .................................. 70
7.2 Case 2 – Technical Lead’s Perception of Same Project ................................................ 72
7.3 Case 3 – Project Manager’s Account of Unsuccessful Project ...................................... 72
7.4 Case 4 – Developer’s Account of Unsuccessful Project ................................................ 73
7.5 Case 5 – Same Developer’s Account of Successful Project ......................................... 74
7.6 Case 6 – Developer’s Account of Successful Iterative Development Project ................. 76
7.7 Case 7 – Requirements Analyst’s Account of Successful Project Resulting in
        Marginally Successful Product .............................................................................. 77

8. CONCLUSIONS AND IMPLICATIONS ......................................................................... 79
9. ANNOTATED BIBLIOGRAPHY .................................................................................... 81
10. APPENDIX A – REQUIREMENTS ELICITATION METHODS ........................................ 90
11. APPENDIX B – DATA, PROCESS AND OBJECT MODELING TECHNIQUES ............. 91
12. APPENDIX C – CASE STUDY INTERVIEWS – QUESTIONS AND ANSWERS .......... 93
Table of Figures

Figure 1: Illustration of Brandford’s IDEAL acronym of problem solving and its relationship to requirements analysis ................................................................. 5

Figure 2: Summary of requirements development processes ............................................... 8

Figure 3: Change in customer expectations as a technology matures over time .................. 17

Figure 4: Effect of usability on system architecture, presentation, functionality and overall system requirements and design, modified from Bass, John & Kates ........................................... 22

Figure 5: Kano’s method for charting product functionality against customer satisfaction ........................................... 25

Figure 6: Families of curves representing priorities of requirements on Kano’s chart .................. 25

Figure 7: Karl Wiegers’ conception of requirements development and post-baseline management, modified to include usability considerations ........................................... 28

Figure 8: Benefit of periodic checkpoints of project scope and post-baseline reviews of proposed changes by the Change Control Board (CCB) ........................................... 29

Figure 9: Rizzo’s conception of the U effect, depicting changing customer satisfaction throughout the duration of the software project .................................................................................. 30

Figure 10: Requirements management process following change control modification .................. 33

Figure 11: Illustration of reduced project risk and shorter project schedule in an iterative development project profile as contrasted with the waterfall approach ........................................... 40

Figure 12: Differences in costs of quality planning and quality assurance ........................................... 47

Figure 13: Nielsen’s model of the attributes of system acceptability ........................................... 53

Figure 14: Value of typical vs. time-constrained products over time ........................................... 61

Figure 15: Activity and cost on a project are lowest during the definition/analysis and planning phases. Any errors caught during these phases will be significantly easier and cheaper to fix than they would be if discovered later in the project and required rework and retesting. Slightly modified from Haynes ........................................... 62

Figure 16: Relationship between defect rate and development time. Projects that achieve the lowest defect rates also tend to achieve the shortest schedules ........................................... 63

Figure 17: Mass’s and Berkson’s conception of the profit impact of quality gates based on their timing during the project. The earlier in the project the quality gate takes place, the greater the pretax profits ........................................... 64

Figure 18: Once requirements are baselined, no project should be planned or implemented unless the project manager has assessed that the organization has adequate resources to meet the requirements ........................................... 69
# Table of Tables

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Activities involved in each phase of requirements analysis</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2</td>
<td>Characteristics of a sound requirement statements</td>
<td>9</td>
</tr>
<tr>
<td>Table 3</td>
<td>Characteristics of sound a requirement document</td>
<td>9</td>
</tr>
<tr>
<td>Table 4</td>
<td>Risks of inadequate requirements processes</td>
<td>10</td>
</tr>
<tr>
<td>Table 5</td>
<td>Usability Benefits Hierarchy</td>
<td>19</td>
</tr>
<tr>
<td>Table 6</td>
<td>User, task and functional analysis steps a requirements analyst can perform to mitigate insufficient usability data</td>
<td>20</td>
</tr>
<tr>
<td>Table 7</td>
<td>Common usability scenarios that require architectural considerations in order to be implemented properly</td>
<td>22</td>
</tr>
<tr>
<td>Table 8</td>
<td>An example of simple prioritization of requirements</td>
<td>23</td>
</tr>
<tr>
<td>Table 9</td>
<td>Characterizations of families of requirements curves on Kano’s chart</td>
<td>26</td>
</tr>
<tr>
<td>Table 10</td>
<td>Effect of Kano’s curve families of requirements on decisions concerning project funding, risk and follow-up interview question styles</td>
<td>27</td>
</tr>
<tr>
<td>Table 11</td>
<td>Requirements Management Activities</td>
<td>32</td>
</tr>
<tr>
<td>Table 12</td>
<td>Project stakeholder roles and responsibilities on requirements change control board</td>
<td>34</td>
</tr>
<tr>
<td>Table 13</td>
<td>Affect of requirements changes on other areas of the project</td>
<td>35</td>
</tr>
<tr>
<td>Table 14</td>
<td>Examples of “lightweight” development methodologies</td>
<td>38</td>
</tr>
<tr>
<td>Table 15</td>
<td>Iterative process resource allocations for more balance, less scrap and rework</td>
<td>40</td>
</tr>
<tr>
<td>Table 16</td>
<td>Summary of classic mistakes that lead to slow development. Ones in bold are ones that sound requirements analysis can alleviate</td>
<td>42</td>
</tr>
<tr>
<td>Table 17</td>
<td>Practices of “back to basics” projects</td>
<td>43</td>
</tr>
<tr>
<td>Table 18</td>
<td>Cause factors leading to successful project, with requirements-related causes in bold</td>
<td>45</td>
</tr>
<tr>
<td>Table 19</td>
<td>Cause factors leading to challenged projects, with requirements-related causes in bold</td>
<td>45</td>
</tr>
<tr>
<td>Table 20</td>
<td>Cause factors leading to impaired projects, with requirements-related causes in bold</td>
<td>45</td>
</tr>
<tr>
<td>Table 21</td>
<td>Quality Planning Core Processes</td>
<td>48</td>
</tr>
<tr>
<td>Table 22</td>
<td>Quality Planning Facilitating Processes</td>
<td>48</td>
</tr>
<tr>
<td>Table 23</td>
<td>KLCl list of “Eight Key Questions” that span the spectrum of Product/Business Planning needs</td>
<td>54</td>
</tr>
<tr>
<td>Table 24</td>
<td>O’Neill’s Characteristics of “old style” project management behaviors</td>
<td>60</td>
</tr>
<tr>
<td>Table 25</td>
<td>O’Neill’s Characteristics of “newly emerging” project management behaviors</td>
<td>60</td>
</tr>
<tr>
<td>Table 26</td>
<td>Kulik’s and Samuelson’s conception of the differences in project activities between “traditional projects” and “e-projects”</td>
<td>66</td>
</tr>
<tr>
<td>Table 27</td>
<td>Capers Jones twelve essential attributes of successful software projects</td>
<td>70</td>
</tr>
</tbody>
</table>
1. Abstract

This paper will demonstrate that the general project management principles' regarding requirements analysis also hold true for software development projects. According to conventional project management wisdom, sound requirements analysis and scope definition tends to improve quality planning, thereby reducing project cost and duration, increasing project success and improving the quality of the resulting product. This paper will demonstrate that software development projects tend to challenge this time-proven notion. The paper will also demonstrate that the software development industry pays a high price for these practices by suffering longer project schedules, higher costs and producing poorer quality products by rushing requirements definition and analysis. The practice of unwise attempts to shorten software projects takes away from both the successes of the project and the quality of the resulting product. This demonstration will be accomplished by means of a literature review and an informal survey of various members of the software development industry.

2. Introduction

Like any product, to be marketable, software must be seen as a solution to a perceived problem. To be saleable, the software product must be perceived as a valid solution. To be truly successful, the product must be perceived as a valid solution that is superior to its competition, and the perceived problem must be valid.

Software products are intangible, but they are at least as complex as any of their physical counterparts, such as a building or a machine. The construction of software requires considerations similar to those that go into the construction of a complex physical object – mastering complexity, organizing and structuring the descriptions of the machine to maximize clarity, and minimizing unnecessary interaction. However, because software is based on an idea for which there is not yet a solution, engineering a software product is even more complex. It is analogous to building a description of a physical product, rather than the product itself.

1 Examples of generic project management principles may be found in many standard project management textbooks, such as the Project Management Book of Knowledge published by the Project Management Institute.

Furthermore, the functionality of software products tends to be less mature than that of physical products. The combination of fewer boundaries and unchartered territory creates many more opportunities to make false assumptions that are not recognized or corrected until the product is already built or even released.

The key difference between engineering a software product and a hardware product is the greater degree to which the intangible product must be conceptualized before it is built. The first step to accomplishing this is clearly defining and understanding the user’s problem. Unlike a bridge or a building, software is an idea, and is therefore relatively free from physical constraints of the real world. Computer logic offers many more possibilities for perfect solutions, along with many more possibilities for bad ones. While avoiding bad solutions is important in the physical world as well, it is much harder to accomplish when the number of possible solutions is greater. One of the key ways to avoid choosing a bad solution is by clearly defining and understanding the problem to be solved. The definition of the problem is the first and the most critical step of requirements analysis.

This paper will analyze the effect of requirements analysis on quality planning, project success, and product success in software development. The idea for this topic was born by many informal personal observations that the management of software development projects, particularly of time-constrained applications, tends to violate some basic principles of project management in the interest of reducing time to market. Definition/analysis is the one phase of the software project that tends to suffer the most. This approach is contrary to traditional project management wisdom that upholds sound requirements analysis, planning, and the reduction of project risk as some of the most important ways to promote project success in terms of cost and timely deliverables. Unforeseen defects discovered late in the project cycle present a high degree or risk to the project. According to the Project Management Book of Knowledge, the first step to risk management is risk identification. Foreknowing impending risks is nearly impossible without a thorough understanding of the project scope, which is based on a thorough understanding of the requirements. Without a risk mitigation strategy, the project is more of an exploratory mission where risks are identified on the fly, and the results are unpredictable.

3 In addition to personal experience, this is evidenced by the results of a 1995 Standish Group Study and by Steve McConnell’s list of “classic mistakes” that cause projects to exceed time and resource constraints, or even to fail. See Sections 4.10 and 4.11 below.

Software development projects are frequently time-driven, and project managers are often put in the unenviable position of having to reduce the time to market. Combined with the common misconception that a software project does not truly start until coding begins, this notion leads to shortening, or even eliminating the preliminary analysis and planning for the project. Presumably, software project managers do this because they believe this practice will increase the success of the software project, with only a minimal effect on the product’s quality.

A literature review suggests that this practice of squeezing out the definition/analysis phase not only falls short of the expected outcomes, but actually has the opposite effect. Reducing the analysis phase tends to reduce probability of successful project completion, lengthens the duration of the software project, increases its cost, and/or results in an inferior software product. Software projects have a notoriously poor track record of on-time completion, and software products have a relatively short market lifespan.

In summary, this paper will accomplish the following:

- Describe and motivate best practices in requirements definition and analysis.
- Demonstrate that a large number of IT project managers attempt to shorten projects by rushing the definition/analysis phase, and consequently are not successful in meeting schedules and budgets.
- Demonstrate that shortening the analysis phase introduces unnecessary risk into the project, and has the effect of lengthening the project duration and producing inferior quality software products.
- Cite case studies to examine success and failure factors in software projects.

3. Methodology

This paper will analyze the effect of requirements analysis on quality planning, project success, and product success. This will be accomplished with a literature review of current practices and compare them with recommendations for 'best practices' in requirements analysis, quality management, project management and product management. In addition, an informal questionnaire of software professionals about projects they recently worked on will be used to collect some data, and to elicit case studies of successful and unsuccessful software projects.
The results will be used to analyze the relationship between requirements analysis and quality management, software project success and product success.

4. Requirements Analysis

4.1 Requirements Analysis Defined

Succinctly put, requirements analysis is important because "If you do not have the correct requirements, you cannot design or build the correct product, and consequently the product does not enable the users to do their work."\(^5\)

A survey of references on requirements analysis suggests that solving the wrong problem is one of the most common errors in software development. This error accounts for a huge number of unprofitable or unusable software products that has been produced in the past, and continues to this day. The most prevalent cause of this error is poor understanding of the problem to be solved by the impending software product.

In addition to the cost of wasted time in attempting to solve the wrong problem, IT companies are increasingly subject to law suits against systems outsourcers and integrators for systems that are not delivered according to contracted commitments. According to Tom DeMarco, a well-known IT consultant who frequently serves as an expert witness in IT-related law suits, "For the typical contract IT operation, the ongoing cost of litigation is part of the budget."\(^6\) Wayne Bennett, a Boston-based attorney at Bingham Dana who often handles major IT contracts, further indicates that "In addition to money paid to vendors, integrators and consultants, ... the major components of an IT failure are time ('You get two-thirds through a project and find you've wasted 18 months working on the wrong solution,' he says) and opportunity cost—that is, the potential competitive advantage you've blown by not working on the right solution.'"\(^7\)

The first and most important contribution of requirements analysis to a software project is the definition of the customers' or users' problem. Once the problem is defined, the need for an

---


automated solution must be justified. If an automated solution is in order, the best one should be chosen from a set of possible solutions. Without a clear understanding of the problem, the tasks of formulating and prioritizing solutions are likely to lead to wrong conclusions. Wrong conclusions result in unnecessary solutions to wrong or non-existent problems. The resulting software products are unmarketable and/or unusable.

Although there are some differences, the common element to any definition of requirements analysis is the understanding of the user problem that the proposed software project aims to resolve. As an example, Peter Horan defines requirements analysis in terms of problem solving, and indicates that problem solving may be viewed as a sequence of steps. A neat acronym, attributed to Bransford\(^8\), outlines five steps involved in solving a problem, the first two of which — identify the problem and define the problem — constitute requirements analysis, as shown in the diagram below, while the last three usually involve requirements management.

Figure 1: Illustration of Bransford's IDEAL acronym of problem solving and its relationship to requirements analysis.

Karl E. Wiegers defines requirements as “a specification of what should be implemented. [Requirements] are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.”\(^9\) Ralph Young describes the goals of requirements analysis as identifying incorrectly elicited assumptions, ensuring consistency, increasing compliance, reducing misunderstandings between organizations and individuals, improving the responsiveness of suppliers, improving the satisfaction of all customers, writing good requirements, and emerging the real requirements.\(^10\)

\(^8\) Bransford, J. D. and Stein, B. S., The IDEAL Problem Solver, Freeman, 1984.


Requirements analysis is frequently confused with design. A key distinction between requirements and design is the way the specifications are written. Where system design specifies *how* the system should work, system requirements specify *what* the system should do by focusing on system functions as they relate to the customers’ and users’ problems.

4.2 **Best Practices in Requirements Analysis**

4.2.1 **General Overview**

Ideally, the system functional requirements should be derived from true user and business needs, without regard to the capabilities of the software organization tasked with meeting the requirements. In the real world, this is often not practical, and system requirements are usually pre-censored before they are written down in order to ensure that they can be met. Requirements are adulterated further by technical stakeholders prior to sign-off. This practice helps to ensure that the projects that a software organization undertakes are realistic. However, it is important also to ensure that all user and business needs are conceived first, and, if necessary, removed from scope later, consciously and deliberately.

Karl E. Wiegers underscores this principle by outlining a sequence of best practices in requirements development as elicitation, analysis, specification, and verification. Activities involved in each phase of requirements analysis are outlined in the table below.
Table 1: Activities involved in each phase of requirements analysis

<table>
<thead>
<tr>
<th>Elicitation/Definition</th>
<th>Analysis</th>
<th>Specification</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define product business requirements (scope)</td>
<td>Decompose high-level requirements into detailed functional requirements</td>
<td>Write down requirements in an accepted, structured format during elicitation and analysis</td>
<td>Evaluate correctness</td>
</tr>
<tr>
<td>Get extensive user involvement (“product champion”)</td>
<td>Constructing graphical requirements models (alternate view of requirements)</td>
<td>Communicate shared understanding of the new product among all project stakeholders</td>
<td>Evaluate completeness</td>
</tr>
<tr>
<td>Focus on user tasks (use cases)</td>
<td>Build prototypes (alternate view of requirements)</td>
<td>Store requirements in a requirements management tool (not a substitute for well-written requirements)</td>
<td>Evaluate usability</td>
</tr>
<tr>
<td>Define quality attributes (acceptance criteria; how well will system perform its functions)</td>
<td>Prioritize requirements</td>
<td>Inspect requirements with key project stakeholders (get sign-off)</td>
<td>Attain common understanding</td>
</tr>
<tr>
<td></td>
<td>Translate into functional requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>State acceptance criteria</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wiegers includes requirements definition as part of elicitation. While the two activities are related, definition of requirements is not based solely on elicitation. Although defined requirements are likely to resemble the requirements elicited from business representatives and users, they also should include inputs from the requirements analyst’s own experience and system knowledge, as well as any data from usability studies, and information about user behavior inferred from use case scenarios. Studying use cases, usability data, and even looking deeper into the business data may reveal that the initially-conceived system is not useful in solving the user’s problem. Even at a high level, defined requirements may look quite different than the requirements elicited from the business or the users directly. Of course if this is the case, it is the requirements analyst’s job to “sell” the defined requirements back to the business, the users and the project stakeholders as their “actual” needs, or advise the project manager that the project should not be undertaken. Not doing so would be dismissing advance notice that the resulting product will not meet users’ expectations.

In summary, the requirements development and management follows this type of process:

---


12 See “Usefulness” section below for additional detail.
Examples of these general principles are echoed in a number of literature sources. Constance Heitmeyer, for example, provides a general description of the SCR (software cost reduction) requirements method, used primarily in high assurance systems, which include command and control systems, weapons systems, flight programs for commercial and military aircraft, control systems for nuclear plants, and most medical systems. High assurance systems are characterized by service properties, security properties, safety properties, real-time properties, and fault-tolerance properties.  

Although the focus of the article is on a specific tool, the basic functions of this tool support general principles of best practice software requirements analysis. The functions include a specification editor, a dependency modeler, a consistency checker, a product simulator, and a verifier of requirements.

Herlea’s article is about the requirements elicitation process, and the importance of involving end-users during this crucial phase of the software project. Although this article focuses on a groupware system called TeamRooms, designed to facilitate communication between participants of a distributed requirements analysis team, the article also describes a number of basic methods of eliciting requirements.


14 Daniela Elena Herlea (Knowledge Science Institute, University of Calgary), “Users' Involvement in the Requirements Engineering Process”, http://spuds.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html, last viewed in April 2001. Not able to regain access to site as of 5/25/01
Herlea describes software requirements engineering as "the process of determining what is to be produced in a software system. In developing a complex software system, the requirements engineering process has the widely recognized goal of determining the needs for, and the intended external behavior, of a system design. This process is regarded as one of the most important parts of building a software system." Furthermore, she quotes another source as follows: "No part of the work so cripples the resulting systems if done wrong. No other part is more difficult to rectify later"\(^\text{15}\)

Like Weigers and a number of other experts, Herlea describes the four critical steps in software requirements engineering as elicitation, analysis, specification, and validation. The difference in Herlea's view is that the steps are not intended to be sequential. Herlea sees them as inter-dependent and iterative, as requirements are expected to change not only throughout the analysis phase, but also throughout and after development. Wiegers implies this idea as well in the requirements management portion of his activities. The process is indeed iterative since each time a new requirements change is proposed, it must be re-elicited, re-analyzed, re-specified and re-validated.

When is requirements analysis complete? According to Karl Wiegers, it is when each requirement statement and the requirements document can be characterized as outlined in the table below.\(^\text{16}\)

<table>
<thead>
<tr>
<th>Table 2: Characteristics of a sound requirement statements</th>
<th>Table 3: Characteristics of sound a requirement document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Correct</td>
<td>Consistent</td>
</tr>
<tr>
<td>Feasible</td>
<td>Modifiable</td>
</tr>
<tr>
<td>Necessary</td>
<td>Traceable</td>
</tr>
<tr>
<td>Prioritized</td>
<td></td>
</tr>
<tr>
<td>Unambiguous</td>
<td></td>
</tr>
<tr>
<td>Verifiable</td>
<td></td>
</tr>
</tbody>
</table>

In addition to the benefits of sound requirements analysis, Karl Wiegers outlines some risks from inadequate requirements processes.\(^\text{17}\)

---

\(^{15}\) In Hearlea, [http://spuds.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html](http://spuds.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html), attributed to Brooks, 1987. Not able to regain access the site as of 5/25/01

4.2.2 Customer Involvement

Herlea emphasizes the negotiation of requirements as one of the key processes in eliciting requirements. In addition to the requirements analyst, the three key participant stakeholders during requirements negotiations are the developer, the user and the customer. The distinction between the user and the customer is not trivial. It is not unusual for the customer and the user to be two distinct parties with different, even conflicting interests. Many times, the customer – or the person who pays for the software product – is not the user of that product. For example, a parent may be the customer while his child is the user. The company’s business is to sell the usefulness of the product to the parent, although the child may find the product uninteresting or hard to learn, and may refuse to use it. An example commonplace in a business software setting is a manager making a software purchasing decision for her subordinates, or a corporate department tasked with making decisions concerning an enterprise-wide MIS system. A corporation may have an online training unit that is in charge or shopping for and selecting an online training product that is expected to meet the needs of the entire organization. The importance of both usability and usefulness is critical, and is discussed elsewhere in this paper.

It seems intuitive that an organization seeking to make a profit by selling a product to customers would be interested in involving customers in product development. Unfortunately, the software development industry does not have a strong tradition of customer engagement. The industry wastes millions of dollars annually by launching software products that do not meet customer

---

expectations, primarily due to insufficient customer involvement during the development process.\textsuperscript{18}

Software developers tend to be regarded as magicians by customers who do not have a good understanding of their technical skills. Often, this affects the requirements specifications process as customers willingly delegate requirements definition to the perceived "experts".\textsuperscript{19} This is a problem because the developers’ focus is often in conflict with the customers’ expectations.

By nature and training, computer programmers are solution-focused. Left to their own devices, the better developers will try to frame the problem in ways to yield the most interesting or elegant solution. The weaker developers will try to avoid difficulty by redefining the problem to yield the least amount of development work. Note that this is not the same as yielding a simpler solution, as advocated by some software development methodologies, just an easier one. In most situations, redefining the problem to fit a desired solution is inappropriate. It is the requirements analyst’s job to ensure that the customer’s true problem is stated in terms that are not biased by a solution. The same is true for defining business and market drivers, as well as human factors issues. A truly customer-focused organization will give no thought to a possible solution until the problem is clearly defined and understood by all stakeholders.\textsuperscript{20}

Although the problem should be stated without regard to the organization’s technical and financial capabilities, no software project should be undertaken without reasonable expectation that it can be completed within the boundaries of organizational expertise. Proceeding otherwise would be a recipe for an unsuccessful project. However, the project must be business-driven first, customer-driven second, and system- or organization-driven third. This cannot be accomplished without a clear statement and understanding of the problem in terms of customers’ needs or expectations. If the organization’s expertise becomes the limiting factor, the project should not be undertaken until the expertise is sufficiently reinforced.

\textsuperscript{18} Shoppel, Michael & Davis, Philip, \textit{The Five Secrets of a Successful Launch}. White paper from BetaSphere, Inc. \url{http://i.nl02.net/beta0006/data/beta_white_sosl.pdf}, last viewed on 5/15/01.


\textsuperscript{20} From personal experience, problem statements from project stakeholders tend to come down in terms of functional specifications rather than business or user needs to be explored and solved by professionals. A number of literature sources also describe the opposite scenario where the users don’t know what they want and are happy to relegate the definition of the problem to the software project team. Either situation poses a danger as to whether the final product is perceived as successful or unsuccessful, and calls for a clear mutual understanding of the problem to be solved and common expectations on the part of all stakeholders.
Another frequent software project pitfall that alienates customers is the encroachment of technical considerations and false assumptions into requirements by requirements analysts and software engineers during product development. Software engineers do this for a number of reasons. They might simply want to be included in decision making, they might be tempted to simplify or complicate the solution, or they might have political motives for more power within the organization. Whatever the reason, anti-customer behavior is best tempered and discouraged. The behavior is especially damaging if it falls outside the established requirements change control process. It is the project manager’s and the requirements analyst’s job to establish proper lines of communication and trust with software engineers to encourage them to willingly express their concerns and assumptions rather than build them into the product and then fail quality tests or not meet customer expectations. Iterative development is an effective way of exposing false assumptions early in the project and to encourage communication between all stakeholders.

4.2.3 User Involvement

As already established, both customer and user involvement is of utmost importance in requirements analysis. The two roles are often distinct, as the customer is not always the user, and the user is not always the customer. From a business perspective, the customer’s input is more important than the user’s. In terms of product quality, however, the user is the most important consideration in defining the product’s functional requirements. In many cases, even if the user is not a paying customer, it is still the user who determines whether or not the product will succeed in the long-term. This section describes the various techniques of involving the user in requirements analysis and in the software development project.

Herlea suggests that a typical requirements elicitation team should consist of a facilitator, the users, the analyst and members of the design team. The facilitator acts as a chairperson of the meeting and has a critical role in organizing the work of the requirements negotiation team. The present or future users are the "owners" of the problem. The analyst represents the design team and has a key role in the transfer of the requirements from the "problem owners" to the design team. The design team consists of system implementers who will be responsible for meeting the elicited requirements and, hopefully, customer expectations.

The notion of a correlation between direct user involvement in requirements analysis and the success of the respective project is very important.

... direct contact between [users]²² and developers is preferable to indirect contact because it decreases filtering or distortion that may occur. Further more [sic], they observe a strong correlation between the number of links used and the relative success of the projects initiated. The primary difference between various user involvement methodologies and requirements engineering techniques lies in the degree to which the users participate in the emerging system.²³

Herlea classifies user involvement into consultative design, representative design and consensus design. Consultative design leaves decision-making power to information system staff. Representative design involves selected user representatives in the actual design formulation and decision making. Consensus design fully shares responsibility for the system development process between the users and developers. Of further interest, is Herlea’s outline and brief descriptions of some recognized requirements engineering methodologies, outlined in Appendix A.

4.2.4 Requirements Validation Techniques

4.2.4.1 Data, Process and Object Analysis

Once business and user requirements are clarified, they are analyzed and translated into functional system requirements through the use of data, process and object analysis and modeling techniques. These techniques are well accepted and are likely to be employed by developers even if a requirements analyst is not involved. In fact, a number of development shops erroneously consider this design rather than requirements analysis. This point of view is in error because data, process and object analysis techniques are useful not only in initiating design, database structure and programming logic, but they also serve to validate business and user requirements as they were initially elicited. In fact, they frequently serve as a source of additional requirements that were missed during elicitation. Even more importantly, they tend to lead users and customers into reconsidering their original ideas as at this point, these ideas take on shape and become more tangible. It is very important to maintain a high level of flexibility

²² The word “customer” is used here, but the rest of the passage uses the word “user” interchangeably with the word “customer”. This is in conflict with Herlea’s deliberate distinction between “user” and “customer” earlier in the article. However, a different source is quoted here – a study by Carmel, Whitaker and George, 1993.

²³ In Hearlea, http://spuds.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html, attributed to a study by Carmel, Whitaker and George, 1993. Not able to regain access the site as of 5/25/01.
and user/customer involvement at this stage of analysis since data, process and object analysis methodologies often help elicit additional requirements, and to surface any hidden faulty assumptions. Appendix B outlines a number of techniques used for data, process and object analysis.

4.2.4.2 Prototyping

Prototyping is another important technique for validating user or customer requirements. Even more so than data and process analysis, prototyping forces users into visualizing the resulting system. As a result, users better understand whether and how their problem is proposed to be solved before time and expense is invested into the development effort. Paper prototyping is a particularly inexpensive way to validate requirements, while also exploring usability issues that must be taken into consideration during user interface design. Prototyping can also be accomplished by producing an electronic mock-up of screens based on the initial functional requirements, using an application as simple as MS PowerPoint, or a set of interconnected HTML pages with no logic behind them other than navigation.

Prototyping has a number of similarities to data, process and object analysis techniques. Like these analysis techniques, prototyping serves as a useful exercise for validating user and customer requirements. Also like the analysis techniques, it tends to get confused for design. And, like data, process and object analysis, it is perceived as dispensable in time-challenged projects. While it is true that “full-strength” prototypes can be time-consuming and expensive to produce, there are ways to prototype that are much less expensive and involved, and produce results that are also useful. A case study of paper prototyping is cited in the Usability section below.

Although requirements validation is not the primary purpose of prototyping, the two activities are highly inter-related. Discoveries from prototyping are bound to uncover new functional requirements. In addition, the prototype is bound to challenge and refine the initial set of requirements on which it is based.

If the software project is conducted using the iterative development approach, then the product is prototyped and tested with each demonstration to the user community. As users explore the prototype, they pinpoint possible misunderstanding of requirements, discover possible additional needs, and reveal usability issues that can then be folded into re-design, or perhaps even a modification of the original business or user requirements.
4.3 Usefulness and Usability Analysis

4.3.1 Usefulness

The idea of "usefulness" may apply to an entire software system, or any part of it. Assuming as a foregone conclusion that the problem requires an automated solution may lead to a useless software product. Perceived user needs for automation that may be identified during requirements elicitation, may be better met without automation. However, the absence or lack of usefulness is usually not a mandate not to build the product. In selling a product, perceived usefulness is often more conducive to short-term sales than actual usefulness. For example, millions of people periodically give or throw away brand new clothes that they loved in the store, but never wore. Consumers buy expensive sports cars with powerful engines, but never exceed the posted speed limit. Consumers buy lottery tickets knowing that their return on investment would be much higher if they saved their money.

This applies to software as well. Very few people are expert users of MS Word, so the majority of the more cryptic advanced features are not used by most users. Yet almost everyone perceives these features as valuable at purchase time, and is usually willing to pay more for them. It is the requirements analyst's and/or the usability expert's responsibility to gather as much usefulness data as possible, and to present it to the project stakeholders for an informed business decision. The useless feature or system may still be deemed worthy of building if it is expected to generate revenue.

Like requirements analysis, usefulness data is best not relegated to professionals with financial and job security interests in selling automated solutions. As with most professionals, programmers will try to help by solving a problem in terms of the discipline they know best. For example, a user may have a perceived need to have his shoelaces tied. If he asks a doctor for help, the doctor will examine the user medically to see what's preventing him from accomplishing his goal, and will probably prescribe a medical treatment. A lawyer will tend to counsel the user to sue the shoe manufacturer for damages if case he trips on a shoelace and breaks his arm. A teacher will be inclined to teach him how to tie a shoelace.

Because programmers are trained to provide automated solutions to problems posed by others, true to their training and profession, they will tend to start with an assumption that the perceived problem is real and that it requires their help, and therefore automation, in order to be resolved.
It is the author’s experience that programmers tend take a system-driven approach to problem solving, by coming up with automated solutions whether or not one is needed, and even inventing a problem to suit a particularly “cool” solution. A requirements analyst or a usability expert is trained in defining and validating user problems. In an environment where development resources are scarce and the number of potential projects is high, this is the best qualified person to analyze the usefulness of an automated solution.

Jakob Nielsen views usability as a derivative of usefulness.\footnote{Jakob Nielsen, \textit{Usability Engineering}. Boston: AP Press, 1993. Page 25.} In other words, in order for a system or its function to be usable, it must first be useful.

\subsection*{4.3.2 Usability}

\subsubsection*{4.3.2.1 Usability Studies}

Although user involvement is a necessary component of sound requirements analysis, it is only one part of defining usability requirements. Just asking users what they want does not constitute a set of usability requirements. As a number of studies and personal testimonies have revealed, users frequently don’t know what they want until they have seen a prototype or the finished product, and have had the opportunity to test it. It would however, be wrong to exclude user input from the development process on this basis, as that is likely to lead to a number of other unwanted problems. The problems are particularly grave to the software development business if the user is also the customer.

The phenomenon of the user not knowing or understanding what they are asking for is even more complex because what users think they want frequently shapes their purchasing decisions. Since selling a product, either externally or internally within the company, is the ultimate goal of any commercial software development organization, it is likely that business requirements will be based on the average customer’s observed purchasing behavior. A software manufacturer may know from market studies that customers are willing to pay a lot for a certain feature, while a usability study might reveal that most users will not be savvy enough to use that feature, or wouldn’t really find it useful even if they could learn the software.

Business requirements are often in conflict with usability requirements. In a corporate setting, business considerations frequently supercede usability considerations, sometimes for sound business reasons. However, a sound business decision is an informed decision. If a product is
predicted to bring in $1,000,000 in the short-term because of its perceived value, but then turn away return customers because of disappointment with the usability of the first product they purchased, business sense may still dictate that investment of time and resources to build the product is warranted. If the business comes to a decision in favor of producing such software with a clear understanding of the risks to the company in the long-term, the business decision may still be fairly assessed as sound.

A number of usability experts acknowledge some legitimacy in the business interests that may run contrary to the usability arguments. For example, Donald Norman observes that in the early stages of a technology, users are satisfied with technology and performance, and are less focused on ease-of-use convenience. However, Norman observes that this changes as the technology matures and more competitors enter the market, user considerations and preferences change in favor of convenience and solutions. He suggests that the relationship of the product’s maturity to the types of customers it attracts looks like this:

Figure 3: Change in customer expectations as a technology matures over time.

![Diagram](http://www.nngroup.com/reports/life_cycle_of_tech.html)

Norman provides the following explanation for this change in customers as technology matures:

In the early days, the innovators and technology enthusiasts drive the market; they demand technology. In the later days, the pragmatists and conservatives dominate; they want solutions and convenience. Note that although the innovators and early adopters drive the

technology markets, they are really only a small percentage of the market; the big market is with the pragmatists and the conservatives. (Modified from Moore [1995]26).

Norman further notes that the fact that late adopters to technology are very different from the early adopters is widely known among technology marketers, who nevertheless seldom change their approach to designing software to accommodate the late adopters. Norman blames this on the fact that "the whole culture of the company is based on its wildly successful teen age years, and high-tech companies hate to grow up. immaturity [sic] is embedded in the culture. Technology is easy to change. Culture is hard. [sic]"27

4.3.2.2 Usability Requirements Analysis

If the project stakeholders agree on the importance of usability considerations in building software, and are willing to allocate time and resources to collect and study usability data, the next step is understanding exactly what is meant by usability. Learnability, efficiency, memorability, errors and satisfaction are the five attributes of good usability design listed by Jakob Nielsen. In other words, the system should be easy to learn so that the user can rapidly start getting some work done. The system should be efficient to use, so that once the user has learned the system, he can be highly productive. The system should be easy to remember, so that the casual user is able to return to the system after a prolonged absence and not have to re-learn everything. Furthermore, the system should have a low error rate, so that users make few errors while using the system, and easily recover from any missteps. The system should be free of catastrophic errors. Finally, the system should be pleasant enough to use, so that users are subjectively satisfied.28

Bass, John and Kates provide a hierarchy of usability benefits in the following chart outlining the benefits of usability.29 The chart can be used to qualify the benefits of each usability recommendation made in designing a software product or feature.


Table 5: Usability Benefits Hierarchy

<table>
<thead>
<tr>
<th>Increases individual user effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expedites error-free portion of routine performance</td>
</tr>
<tr>
<td>- Accelerates error-free portion of routine performance</td>
</tr>
<tr>
<td>- Reduces the impact of routine user errors (slips)</td>
</tr>
<tr>
<td>Improves non-routine performance</td>
</tr>
<tr>
<td>- Supports problem-solving</td>
</tr>
<tr>
<td>- Facilitates learning</td>
</tr>
<tr>
<td>Reduces the impact of user errors caused by lack of knowledge (mistakes)</td>
</tr>
<tr>
<td>- Prevents mistakes</td>
</tr>
<tr>
<td>- Accommodates mistakes</td>
</tr>
<tr>
<td>Reduces the impact of system errors</td>
</tr>
<tr>
<td>- Prevents system errors</td>
</tr>
<tr>
<td>- Tolerates system errors</td>
</tr>
<tr>
<td>Increases user confidence and comfort</td>
</tr>
</tbody>
</table>

Although these ideas are intuitive to most software users, they are frequently overlooked in analysis and not administered in design of the software development projects. Depending on the software organization, the requirements analyst may not be a usability expert, and may not have access to a usability expert. Even if an organization takes the trouble to hire a usability expert, it may not have the will or the resources to support the usability expert’s efforts. A usability expert is only as valuable as the user data available to him. Usability studies can be brief and inexpensive, or they may be lengthy, involved and costly. Without these studies, however, a usability expert’s input to the functional requirements and interface design is of very limited benefit to the organization.

In the absence of usability studies, the requirements analyst still has a number of options in performing requirements analysis, to bring out usability issues and attempt to address them prior to implementation. The most important way is to understand the user need and translate it into

---

30 The management of one organization I have worked for appeared to accept the need for staff human factors experts, but did not understand the scope of the commitment required for the usability expert to do his or her job. At this company, the usability experts were expected to report to work already possessing all the knowledge necessary to provide consulting support and prepare interaction design deliverables. The company did not understand the need for allocating project time or resources to conduct usability studies or how to integrate their advice with business or systems requirements. The usability experts in this organization were forced by management to produce user interaction designs without a clear understanding of the user base of the future system or product. In addition, the management team failed to convey to the rest of the organization the function of the human factors specialists, which undermined the usability experts’ authority and caused multiple conflicts between the human factors specialists, requirements analysts, business producers and developers. This situation led not only to wasted resources on the part of the organization, but also to severe job dissatisfaction and employee turnover. Anecdotal evidence suggests that unfortunately this is closer to the industry norm than the exception.
system requirements by walking through use cases, task analysis and functional analysis steps outlined in the following table:

Table 6: User, task and functional analysis steps a requirements analyst can perform to mitigate insufficient usability data.

<table>
<thead>
<tr>
<th>User Analysis</th>
<th>Task Analysis</th>
<th>Functional Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify all types of users who expect to be interacting with the system or feature</td>
<td>4. Identify the tasks each user will perform in order to meet each of his needs</td>
<td>7. Identify system functions for each sub-task that calls for interaction with the system</td>
</tr>
<tr>
<td>2. Define and clearly visualize each user type in detail</td>
<td>5. Decompose each task into smaller and smallest sub-tasks</td>
<td>8. Decompose system functions into smallest pieces</td>
</tr>
<tr>
<td>3. Identify the needs of each user type</td>
<td>6. Identify which, if any, sub-tasks will benefit from automation</td>
<td>9. Document functional system requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10. Validate assumptions</td>
</tr>
</tbody>
</table>

The second step is critical in understanding the user’s true needs. It is very useful to develop a detailed user profile, including as many relevant attributes of the user as necessary. These attributes may include the user’s age, professional circumstances, computer savvy, education level, where they will interact with the system (home, office, airplane, etc.), how much time will they have, their attention span, family circumstances, hobbies, and so on. Once functional requirements are identified, they should be validated through the eyes of each user type.

Assumptions should also be tested by means of involving user representatives in a simple paper prototyping session. Because users’ initial requirements frequently don’t correspond to their true needs, prototyping early in the project life cycle has the advantage of validating not only the design, but also of forcing users to reconsider their original requirements early in the development cycle. However, a clear distinction must be made between user requirements and business requirements, as business requirements will usually supercede those of the users. A good example of this is a paper prototyping session where the users express a dislike for entering time into the system in terms of both hours and minutes, as may or may not have been specified by a business requirement. Based on the outcome of the prototyping session it would seem that entering just the hour is sufficient, however modifying the design based on this observation without consulting with the requirements analyst about possible conflicts with business needs can interfere with other users of the system. For example, a user whose job depends on billing reports detailed to the minute, may require data entry in terms of both hours and minutes.31

The results of this approach are significant and worthwhile, but in the absence of a strong advocate for usability requirements analysis, this approach is usually not followed. Typically, the business, customer or user representative will present his needs to the development team, and the development team will immediately start formulating a solution. The requirements analyst documents the expressed requirements without performing any analysis or usability research. The typical programmer will be tempted to visualize herself as the user, and design the system for herself. If the solution seems difficult, the programmer may be tempted to simplify the problem, rather than work out a more suitable solution.

In addition to a negative effect on product success, not considering the user and usability in the definition/analysis phase can have a perilous effect on the success of the project. User issues that are not considered upfront, will usually surface during system development or testing. For example, a tester may realize that the system is missing a navigation mechanism between two highly inter-related views. Another common example is the handling of user error messages. In a client/server or web environment, errors may occur in several places -- on the client side, in a back-end database, during transport through the internet, caused by a server error, etc. Each of these warrants a different type of message and user responses. Ideally, the possible error messages would be forecast during definition/analysis, and their handling would be planned into the project. Without such planning, the system ends up either with poor error handling functionality, or the project completion date is delayed at the last minute.

While some usability issues may be easy to fix on the fly, others may require redesign at the system architecture level and may take significant amounts of rework to fix. Bass, John and Kates consider usability as one of seven possible considerations that go into system design, particularly from a system architecture perspective, as shown in the following diagram:

---

32 This scenario is taken directly from personal experience. On one major web-based system re-engineering project, error-handling specifications did not get early attention from the architecture and development teams, and instead were left for the end of implementation. By the time this issue was revisited, the distinct error messages specified -- client-side vs. server-side errors -- were found to be more difficult to generate because they were not planned for. The error-handling functionality that was implemented was not nearly as helpful to the user as envisioned during requirements elicitation. The resulting error messages were not explicit enough for the average user to understand the cause of the problem or to execute the expected corrective action.

Bass, John and Kates identify a list of 26 common usability scenarios – possible interactions that some stakeholders, such as an end user, a developer, or a system administrator may have with the system from a usability point of view. These scenarios are best handled through design consideration and decisions in the software architecture:

| Table 7: Common usability scenarios that require architectural considerations in order to be implemented properly. |
|---|---|
| 1. Aggregating data | 14. Modifying interfaces |
| 2. Aggregating commands | 15. Supporting multiple activities |
| 3. Canceling commands | 16. Navigating within a single view |
| 4. Using applications concurrently | 17. Observing system state |
| 5. Checking for correctness | 18. Working at the user’s pace |
| 6. Maintaining device independence | 19. Predicting task duration |
| 7. Evaluating the system | 20. Supporting comprehensive searching |
| 9. Retrieving forgotten passwords | 22. Working in an unfamiliar context |
| 10. Providing good help | 23. Verifying resources |
| 11. Reusing information | 24. Operating consistently across views |
| 12. Supporting international use | 25. Making views accessible |

4.4 Requirements Prioritization and Scope Definition

Although the results of the requirements elicitation and analysis methods are valuable, the functional requirements they generate do not automatically constitute the scope of the project. Both the requirements analyst and the project manager must winnow out the requirements that will become part of the imminent project, and those that will be left for future projects. Scope

---

analysis and definition consists of a number of very important considerations. The priority of each requirement to the organization is a first key step.\textsuperscript{35}

There are many simple and complex ways to prioritize requirements. Perhaps the most crude one is labeling each requirement as a "must have" or a "nice-to-have". A more precise method is defining a scale of priority, such as 1 through 4, where 1 is highest priority and 4 is lowest priority, and then assigning a priority number to each requirement. The result may look something like this:

Table 8: An example of simple prioritization of requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement A (&quot;House shall have a roof&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>Requirement B (&quot;Supporting beams shall be at least 6 ft apart.&quot;)</td>
<td>4</td>
</tr>
<tr>
<td>Requirement C (&quot;House shall have windows at least 5 ft wide.&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>Requirement D (&quot;House shall have gold toilets&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>Requirement E (&quot;House shall have pink flamingo on front lawn&quot;)</td>
<td>3</td>
</tr>
<tr>
<td>Requirement F (&quot;House shall have a foundation&quot;)</td>
<td>2</td>
</tr>
</tbody>
</table>

The project manager can then plan the project, with agreement from all stakeholders, based on all requirements of, say, priority 1 through 3. This prioritization scheme can be used later to descope the project if it is running behind projected budget or schedule, or, in theory, to add functions if the project is exceeding the projected schedule and has resources to spare.\textsuperscript{36}

Although this process is subjective and somewhat arbitrary, it can be useful if the results are agreed upon by all stakeholders. Also, this type of prioritization scheme is probably the least time-consuming to conduct. One of its drawbacks is that this process can undo the painstaking analysis that went into defining and analyzing requirements. This is because each requirement tends to be assigned a priority number in a vacuum, without assessing the impact of its dependence on other requirements. In reality, some of the requirements may have been derived from others, and may be inter-dependent. For example, in Table 7 above, Requirement B ("Supporting beams shall be at least 6 ft apart.") is a low-priority requirement, but it may be a pre-requisite for Requirement C ("House shall have windows at least 5.5 ft wide."), which is high-priority. Therefore, Requirement B should be a high-priority requirement as well.

\textsuperscript{35} Along with the requirements priority, another important consideration is the project manager's assessment of the organization's resources and capabilities. As discussed elsewhere in this paper, an organization must never presume to undertake a project without first understanding its own capabilities. However, because this issue is outside the realm of requirements analysis, as it is handled by the project manager, it will not be covered here in detail.

\textsuperscript{36} From personal experience, the latter scenario never happens.
However, considered by itself, this requirement may not appear very important. These types of problems are not always easy to discover until implementation time. This problem can be mitigated by clustering mutually dependent requirements so that they may be prioritized as units. The problem can be even more serious, however, if one requirement is a pre-requisite for multiple other mutually unrelated requirements.

A less subjective approach to prioritizing requirements is one that resembles a hockey tournament. Each team in the tournament starts off by playing each of the other teams. Upon completion of the initial round, each team is ranked by the number of games it has won. A set of requirements may be approached similarly. Each individual requirement may be compared to each of the remaining requirements, and be labeled as a “winner”, “loser” or “mutually dependent”. Upon completion of the comparisons, the requirements can be ranked from those that were deemed “winners” the highest number of times, to those that were “winners” to the fewest number of comparisons. The ranking may stop there, or the hockey analogy can be taken further, to the “play-offs”. For example, each 10 can be compared against all other 10s, each 9 against all other 9s, etc., until all the redundancies are resolved. Mutually dependent requirements should be considered as a set. Although this method does not eliminate the danger of missing important interdependencies, it is less likely to permit them because it forces each requirement to be compared against every other. In addition, this procedure increases the probability that a mutual dependency will be noted.

“Kano analysis” is another useful way of categorizing and prioritizing requirements. In addition, it enables the analyst to compare the importance of each functional requirement against other external factors, such as customer satisfaction for each stated requirement. To start, the requirements analyst gathers enough data to be able to project customer satisfaction. Kano’s method involves graphing product functions against projected customer satisfaction, using the following chart:

---

The Effect of Software Requirements Analysis on Project Success and Product Quality

Vera Z. Berenbaum
Information Technology

Figure 5: *Kano's method for charting product functionality against customer satisfaction.*

- **Quadrant Upper Left**
  - Latent requirements
  - Increasing satisfaction with missing or withheld functions

- **Quadrant Upper Right**
  - Customer delight

- **Quadrant Lower Left**
  - Customer dissatisfaction with missing or withheld functions

- **Quadrant Lower Right**
  - Customer dissatisfaction with provided functionality

The resulting curves are categorized into families, each depicting the value of the requirements in the customers' eyes. The four families of curves look something like this\(^{38}\):

Figure 6: *Families of curves representing priorities of requirements on Kano's chart.*

The families of curves are categorized as follows\(^{39}\):

---

Table 9: Characterizations of families of requirements curves on Kano’s chart.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning/Characterization</th>
<th>Effect on Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = “Attractive/ah-ha!”</td>
<td>If these requirements are not present in the product, the customer will not miss them. Features customers do not know they need until they see them.</td>
<td>• High risk</td>
</tr>
<tr>
<td>discriminators</td>
<td></td>
<td>• Not mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Usually not tied to legacy products already in the market</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Often highly leveraged for value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If successful, bring high return on investment</td>
</tr>
<tr>
<td>L = Linear</td>
<td>Customer satisfaction tends to grow with increasing performance or functionality.</td>
<td>• Require constant/repeated investment for refreshment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Impact decays unless there is mandatory continuing improvement to maintain or expand market share.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• High priority for funding due to great influence on customer satisfaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “Set aside” required in the overall product development budget to continue innovation and competitiveness</td>
</tr>
<tr>
<td>I = Indifferent</td>
<td>Satisfied or not, these requirements have no effect on customer satisfaction. Examples include government-regulated safety requirements, or organization-specific internal requirements.</td>
<td>• Make minimum resource investment possible to meet the requirement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Take no unnecessary risks beyond those needed to achieve compliance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Comply, but challenge these requirements before expending time and money</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Look for opportunities to gain customer reward for these requirements thereby moving them to the upper half of the Kano chart.</td>
</tr>
<tr>
<td>M = “Must be present”</td>
<td>If required function is present, customer probably won’t notice, but its absence will be glaring and will cause customer distress. Example: A PC that is missing a hard disk drive.</td>
<td>• Constantly reevaluate risks; take no more risk than required for minimum capability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Regulate resource impact; make minimum commitment possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Because of potentially sharp reaction to missing or dysfunctional performance, the project manager is mindful of obsolescence; minimum effort to keep up the minimum market standard.</td>
</tr>
</tbody>
</table>

Although here, Kano’s tool is cited as a way to prioritize requirements in making funding choices and evaluating project risk, it has a number of other functions. The tool is useful in developing customer and end-user follow-up interview questions in prototype and design reviews. The

The following table provides an example of follow-up interview question styles suggested by the various types of functional requirements:

Table 10: Effect of Kano's curve families of requirements on decisions concerning project funding, risk and follow-up interview question styles.

<table>
<thead>
<tr>
<th>Curve Family</th>
<th>Funding</th>
<th>Risk</th>
<th>Question Style</th>
<th>Testing for:</th>
</tr>
</thead>
</table>
| A = “Attractive/ah-ha!” discriminators | Discretionary investment targeted for high returns | Take all necessary risks to assure success | Confirming a design       | • Is this still an “A”?  
• Does the design conform to needs/requirements?  
• How does the appeal of the function vary by customer segment? |
| A = “Attractive/ah-ha!” discriminators | Initially discretionary investment targeted for high returns | May decay from “A” to “M” quickly; minimize risk to balance rewards | Exploratory              | • Is this still an “A”, or has it become an “M”?  
• What features could promote this functionality back to “A”? |
| L = Linear       | Constant refreshment required; reserve funds to meet needs | Take prudent risks to maintain market acceptance | Exploratory to limits    | • Is this still an “L”?  
• How much more development of this function would make a difference in the buying decision?  
• How does the appeal of the function vary by customer segment? |
| I = Indifferent  | Mandatory funding to meet minimum requirements | Take no risks not essential to meeting compliance specification | Converging to yes/no    | • Any promotional possibilities here?  
• Indifference to meeting requirements |
| M = “Must be present” | Minimize funding to lowest possible cost | Take no risks, especially if functionality is already mature | Take-away and confirming | • Reaction to removing a familiar or expected function  
• Could this trigger a “no buy”? |

4.5 Requirement Change Control Management

4.5.1 Change

Defining requirements and coming to consensus by the project’s stakeholders is one of the first critical steps in baselining the scope of a software project. However, any sense of completion at this point is illusory, in spite of indications that changes in product or project requirements after

---

the initial scope baseline pose a risk to a great majority of software development projects. Furthermore, the later in the project scope changes are identified, the more costly they are to address and implement. If scope changes are permitted, they can add to or invalidate existing requirements. Therefore, unless the requirements document is updated to reflect changes, the value of the requirements document diminishes rapidly. On the other hand, if requirements are not permitted to change, particularly on long-term projects, the value of the product rapidly diminishes during implementation. By the time the product is released, it may be obsolete. A review of the literature almost unanimously supports the fact that requirements updates after the initial scope baseline are unavoidable. Karl Weigers’ view of the requirements development process, the management processes, and the boundary between them is illustrated below:

Figure 7: Karl Wiegers’ conception of requirements development and post-baseline management, modified to include usability considerations.

While usability considerations are a part of the initial requirements elicitation process, they are validated by prototyping after requirements are baselined. The validation efforts may spur rethinking of some requirements, but deference should be given to business considerations and

---

the change control board. For example, the results of the prototyping session described earlier, where a significant number of users express aversion to entering both hours and minutes into a system, preferring to enter only hours, should be sent back to the requirements analyst for business consideration of the suggested design change.42

In addition to jeopardizing product quality and timeliness, not allowing either the business or the project environment side to suggest changes to project scope jeopardizes the success of the project because it promotes a growth of a disparity between customers’ changing expectations and their original requirements. The resulting gap becomes particularly apparent at the end of the project, when its fruits are presented to the customers. The following figure illustrates the importance of requirements change control:

Figure 8: Benefit of periodic checkpoints of project scope and post-baseline reviews of proposed changes by the Change Control Board (CCB).

Another important reason for tolerance of requirements changes is Michael Rizzo’s observation of the “U Effect” throughout the duration of the project. The U Effect depicts changing customer

satisfaction throughout a program or a project. The reality of the U effect underscores the importance of maintaining lines of communication between all project stakeholders, and accepting the fact that requirements and customer expectations are subject to continual change. The U-shaped customer satisfaction curve is shown below:

Figure 9: Rizzo’s conception of the U effect, depicting changing customer satisfaction throughout the duration of the software project.

Rizzo calls the U effect a “natural phenomenon of changing customer satisfaction over the project period”, characterized by “differences in expectations and misunderstandings, as well as ‘challenges’ in the areas of technical requirements, cost, and schedule.” While it may be successfully argued that the “Depth of Despair” can be mitigated with a more in-depth requirements analysis effort prior to plunging into the project, a certain amount of let-down at initial phases of implementation is unavoidable. Continual open communication will remind stakeholders of the value of their original requirements, as well as give them the opportunity to rethink them. Rizzo suggests that the most successful approach is to “manage, understand and communicate expectations on both sides.” In other words, building a common understanding by encouraging the airing of changing expectations throughout the implementation phase, building consensus and documenting the subsequent changes in requirements and project scope.
The U effect phenomenon applies to any combination of customer and software development organizations where the organization supplies the software product. Rizzo attributes part of the problem to the “competitive process and non-perfect specifications” before a supplier is chosen. The “competitive process” scenario may apply to a situation where the software supplier organization belongs to the customer’s own company. For example, there may be a competitive “bidding” process for the project between several internal organizations, or an internal organization may have been competing and won against outsourcing suppliers. Even if the software organization is not facing any competition, there is usually a looming business question of whether one project gets funding over a competing project, or whether a project is undertaken at all. Rizzo’s observation, therefore, applies to almost any software development project that an organization may “choose” to undertake in-house or to outsource.

4.5.2 Control

As discussed, characterizing requirements changes as unwelcome is usually self-defeating. In the ideal world, the project plan will control the project, rather than the other way around. In the real world, however, the number of external forces is so great and so unpredictable, that denning their existence, ignoring them, or addressing them without updating the original requirement document constitutes the beginnings of a project death march. It is best to assume that requirements will be subject to change as soon as they are baselined. No project plan is complete until the project itself is complete.

The most frequently requested changes will tend to increase the scope because, as a rule, the changes tend to add more functionality than they subtract. This phenomenon of creeping requirements happens to be the most serious cause of project delays and failures. McConnell cites studies that blame feature creep on excessive schedule pressure, cost and schedule overruns, and project cancellations.44 However, forbidding requirements changes, particularly in projects of over six months in duration, is less practical as it poses even greater dangers.

The answer is to control the requested changes by adhering to strict change proposal submission processes, rules of conduct during Change Control Board meetings, and most importantly, never agree to accept a change before its impact on the project is evaluated. Furthermore, all projects

stakeholders must be made aware of the additional time, cost or quality trade-offs that will accompany the requested change.

Requirements engineering consists of developing the three major categories of requirements – business, user and functional – and then managing them. Wiegers suggests the following steps in managing of requirements:

<table>
<thead>
<tr>
<th>Table 11: Requirements Management Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate impact of proposed changes</td>
</tr>
<tr>
<td>Trace individual requirements to downstream work products</td>
</tr>
<tr>
<td>Track requirements status during development</td>
</tr>
<tr>
<td>Monitor project status (track percentage of requirements implemented and verified, just implemented, or not yet fully implemented)</td>
</tr>
<tr>
<td>Manage requirements changes (have process for submitting, evaluating, deciding upon, and incorporating into the requirements baseline)</td>
</tr>
<tr>
<td>Establish change control board of decision-makers to approve/reject each proposed change</td>
</tr>
</tbody>
</table>

Managing requirements as a result of change control activities encompasses similar activities to those that immediately follow their definition. The diagram used earlier to illustrate requirement definition/analysis activities still applies to requirements change control and management with minor but important modifications:
4.6 **Best Practices in Requirements Change Control**

The Project Management Book of Knowledge defines project control as "the process of comparing actual performance with planned performance, analyzing variances, evaluating possible alternatives, and taking appropriate corrective action as needed."\(^{45}\) Any changes to the project’s baseline must be approved by a Change Control Board (CCB). The CCB is a formally constituted group of stakeholders responsible for approving or rejecting changes to the project baseline.

Changing project requirements should be minimal and well-communicated to all stakeholders. An organization serious about controlling its software projects will ensure that the proposed changes are carefully evaluated, the appropriate individuals make decisions about changes, the changes are communicated to all affected participants, and that the project incorporates requirements changes in a disciplined fashion. Project scope changes become more expensive and severe as the project becomes older, and that the value of the system requirements document diminishes with each change that is not duly reflected in the requirements document. Therefore,

---

it is very important to update the requirements document each time there is a scope change in the project.

Furthermore, Wiegers points out the importance of evaluating each proposed scope change from the top down. When a change is proposed, no matter how small, the highest-level requirements must be reviewed for consistency with this change before the individual low-level requirements are modified. The consequence of omitting this step is inconsistency between child and respective parent requirements.46

With very few exceptions, the best practice in conducting requirements change control is first securing agreement for each proposed change by a committee of stakeholders comprising the Change Control Board. Although the membership of such a committee will vary from organization to organization, this committee must always include the following stakeholder representatives:

<table>
<thead>
<tr>
<th>Stakeholder role</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager</td>
<td>Responsible for project control and project success, and therefore would have much to gain from accounting for each proposed change. The project manager would provide to the CCB information about how the proposed change would affect project schedule, cost, resources and possibly quality.</td>
</tr>
<tr>
<td>Business representative</td>
<td>Responsible for paying the bills for the project, and will ultimately have to approve any cost over-runs due to scope changes.</td>
</tr>
<tr>
<td>User representative/advocate</td>
<td>Responsible for analyzing the impact of the proposed change to the user, and to advise the CCB on user impact.</td>
</tr>
<tr>
<td>Development representative</td>
<td>At some point, the decision would have to be made whether a proposed change is technologically feasible. A development representative and/or a system architect are ideal people to make this call to the CCB.</td>
</tr>
<tr>
<td>Requirements analyst</td>
<td>In addition to the responsibility of updating the requirements document once this change is made, the requirements analyst must analyze the proposed change for consistency with other requirements, and advise the CCB of any issues.</td>
</tr>
<tr>
<td>Quality tester</td>
<td>Responsible for planning all the necessary tests to make sure any additional functionality eventually performs to quality standards.</td>
</tr>
</tbody>
</table>

Requirements change control comes under the general heading of project control, which includes control in almost all areas of project management47:

46 Karl Wiegers, Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. p. 280

Table 13: Affect of requirements changes on other areas of the project.

<table>
<thead>
<tr>
<th>Project Management Area</th>
<th>Related Change Control Activity</th>
<th>Affected by Requirement Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope Management</td>
<td>Scope change control</td>
<td>✓</td>
</tr>
<tr>
<td>Time Management</td>
<td>Schedule control</td>
<td>✓</td>
</tr>
<tr>
<td>Cost Management</td>
<td>Cost control</td>
<td>✓</td>
</tr>
<tr>
<td>Quality Management</td>
<td>Quality control</td>
<td>✓</td>
</tr>
<tr>
<td>Human Resource Management</td>
<td>Project team control</td>
<td>✓</td>
</tr>
<tr>
<td>Communication Management</td>
<td>Change control board</td>
<td>✓</td>
</tr>
<tr>
<td>Risk Management</td>
<td>Risk monitoring and control</td>
<td>✓</td>
</tr>
<tr>
<td>Procurement Management</td>
<td>Procurement control</td>
<td>✓</td>
</tr>
</tbody>
</table>

4.7 Project-Related Reasons to Update Requirements after the Baseline

Typically, quality assurance test scripts are written against documented requirements. If scope changes result in new requirements and the requirements document is not updated, the added functionality may be tested and any problems with it may not be identified until the product is released. In addition, when new requirements creep into project scope, they may override any originally documented requirements. If documented requirements no longer accurately reflect the actual requirements, the project will unnecessarily fail any quality assurance tests written against those requirements.

Even if a scope change does not result in failed QA tests or untested functionality, project stakeholders may not be satisfied with the outcome of the project because they may not agree with the decisions to change the scope. Any stakeholders that were excluded from the Change Control Board may have serious concerns with scope changes. For example, the reduced functionality may be unacceptable to the business, to the customers, or to the users. The increase in costs or the project completion date may be unacceptable to the business representative sponsoring the project. And finally, political issues of being excluded from decision-making (ruffled feathers) may undermine the organization’s future successes. Updating the requirements document is a very convenient way of documenting agreement on post-baseline changes to project scope. If stakeholders disapprove or are not aware of the changes that they may or may not have authorized, they will deem the project a failure upon its completion.

4.8 Organizational Benefits of Updating Requirements and Documenting System Functionality

While the most immediately painful reasons for updating requirements are failing QA tests, failing to test all functionality, and failing customer satisfaction tests upon completion, there are some long-term practical reasons why an organization should insist on keeping requirements...
documented and updated. It is very useful to the organization to complete each software project with complete system documentation. Up-to-date system requirements serve as a very effective way to document system functionality for future reference. Ideally, the post-implementation requirements would be taken a step further and reorganized in a logical sequence, such as by functional area within a system. For example, if a software project results in changes to three functional areas of a system—e.g., shopping cart, catalog and billing on an e-commerce web site—upon project completion, requirements pertaining to the shopping cart area would be merged and reconciled with requirements from other possible projects that may have affected the shopping cart area. This way, when someone wants to learn about shopping cart functionality of this particular system, all the functions for that area are grouped together.

The benefits of good system documentation are obvious and many. The most obvious one is educating new employees about the functionality of the system and promoting a common understanding. Furthermore, a documented set of system functions is a great tool for analyzing new requirements for future projects. Rather than writing new requirements for an existing system from scratch, existing system functions may be consulted, and better-informed requirements may be written. Too often, requirements for a complex system are generated from customers who are not well versed in system functionality. As many as 60-80\% of system change requests can be addressed by educating users about existing system functionality, thereby greatly reducing the number of unnecessary system changes, duplicate functionality, and development costs.48 Documented system functionality serves as a valuable starting point. Furthermore, if the organization ever chooses to migrate system functionality to another platform, all the requirements are already documented and organized for the convenience of the re-engineering analyst, coders and testers.49

48 Personal experience and anecdotal evidence

49 From personal experience with three system re-engineering projects, these kinds of projects tend to be very chaotic. Although very often they are motivated by perceived inadequacies of the existing system, in reality, stakeholders soon find that they really don’t want to part with most of the functionality of the existing system. In addition, at the point when an old system is being replaced, the organization suffers from particularly poor employee morale and turnover, especially on the part of the employees who have much of their professional past invested in the old system. This is when documentation of requirements for the old system is most missed. Without this knowledge base, the replacement system brings a few improvements, but loses so much of the old functionality that the system is no longer welcome by the time it is finally in production.
Finally, if the numerous benefits of system documentation are not enough of an incentive to include it as a regular part of requirements management, system documentation is a requirement for any organization that hopes to achieve the “repeatable” level of software process maturity, as defined by Capability Maturity Model (CMM) Level II.50

4.9 Alternatives to the Waterfall Model

4.9.1 Extreme Programming, Iterative Development and Requirements Analysis

Extreme Programming is an example of a lightweight software development methodology. The emergence of the lightweight approach to software development serves as evidence of the fact that software development industry is rediscovering the benefits of sound requirements analysis principles, in particular, those having to do with user involvement and requirements change control.

“Lightweight” methodologies are today’s answer to the now-so-called “heavyweight” methodologies, such as the traditional “waterfall” approach to software project management. The heavyweight approach dictates that the project strictly follow a prescribed sequence of activities – requirements gathering, followed by design, followed by coding, followed by integration, followed by testing. A major criticism of the waterfall process is that it tends to increase project risk by not elucidating design flaws until system integration begins. Because many design flaws do not surface until integration time, integration and testing ends up taking up to 40% of the project development effort.51

In their day, “heavyweight” methodologies were developed to impose structure and discipline on software development efforts as a counter-measure to overly complex, buggy and undocumented code emblematic of the early stages of the software development industry. Not surprisingly, “heavyweight” methodologies came to be perceived as expensive because they are fraught with rules, practices, and documents, and require discipline and time to follow correctly. The overhead is cumbersome, time-consuming and can take on a life of its own, thereby subverting the development effort. Another problem with heavyweight methodologies is that they


51 Rational e-Development Company, Improving Software Economics, presented in Rochester, NY by Walker Royce on 6/12/01.
necessitate highly predictable situations where requirements can be written down and frozen for
the remainder of the project. Although such conditions are not impossible to create, in today’s
software market, they are impractical for longer-terms projects – ones slated to take over six
months to complete. The changing market conditions and, to a lesser extent, user requirements,
tend to render a software system obsolete before it is ever implemented, unless requirements are
allowed to change with the times throughout the lifecycle of the project.52

In response, lightweight methodologies have only a few rules and practices, or rules that are easy
to follow. Lightweight methodologies for building software were designed to reduce costs.

<table>
<thead>
<tr>
<th>Lightweight Development Methodologies53</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme Programming (XP)</td>
</tr>
<tr>
<td>Cockburn's Crystal Family</td>
</tr>
<tr>
<td>Open Source</td>
</tr>
<tr>
<td>Highsmith's Adaptive Software Development</td>
</tr>
<tr>
<td>SCRUM</td>
</tr>
<tr>
<td>Coad's Feature Driven Development</td>
</tr>
<tr>
<td>DSDM (Dynamic System Development Method)</td>
</tr>
<tr>
<td>The Agile Alliance</td>
</tr>
<tr>
<td>Rational Unified Process 54</td>
</tr>
</tbody>
</table>

The common element in lightweight programming is very similar to the sound principles of
requirements analysis – adaptive, rather than predictive nature, and their people-first, rather than

52 Personal experience – I served as one of three requirements analysts on a software project, tasked with
compiling all the business and functional requirements. The requirements gathering and sign-off task took
about three months, and the entire project took almost 20 months to complete. By the time the system went
into production, the requirements were 1.5 years old. The system was in production for all of 2 months
before it was dismantled. The main reason for poor customer acceptance of the system was the early
freezing of the business requirements upon their definition and sign-off. The requirements were not allowed
to change throughout the 20-month development cycle. In fact, the requirements analysts were reassigned
to other projects immediately upon completion of the initial requirements documents because changes to
requirements were “not anticipated.” Multiple project delays also contributed to the problem of the
growing gap between original requirements and true business needs.

53 Martin Fowler, The New Methodology.
http://www.martinfowler.com/articles/newMethodology.html#N401, last viewed on 6/1/01.

54 According to Walker Royce, who led the Rational e-Development Company seminar, Improving
Software Economics, in Rochester on 6/12/01, the Rational approach can be regarded as similar to a
lightweight methodology, but is less restrictive in that it works for larger teams (although one of its aims is
to reduce team size) and is more tailored to the needs of the individual software organizations. Extreme
Programming can be viewed as a subset of the Rational Unified Process.
process-first orientation. In essence, this approach is a combination of sound requirements analysis and continual change control throughout the development cycle.

Extreme Programming’s (XP) primary focus is on simplifying code and managing requirements changes during the development cycle. XP also formalizes the fundamental customer engagement principles of sound requirements analysis – communication, feedback and courage. The four values sought by XP programmers are communication, simplicity, feedback and courage. Extreme programmers communicate with their customers and fellow programmers. They keep their design simple and clean. Feedback from testing software starts in implementation. The system is delivered to the customers as early as possible and changes are implemented as suggested. And finally, extreme programmers courageously respond to changing requirements and technology.

The basic idea behind XP is analogous to iterative development or continuous prototyping. The system is started from an architectural foundation and a subset of the initial system requirements, and is brought back to the users for inspection and testing periodically throughout the development cycle. Any corrections due to problems encountered with usability issues, misinterpreted requirements, rethought requirements, changing user needs or market conditions are implemented gradually, reducing project risk and maintaining a level set of expectations between the customers and the developers. In addition to corrections, each iteration of the system includes new features that are gradually added from remaining or new requirements.

In fact, because the iterative development approach welcomes requirements changes and continuous input, it is more adept at addressing usability issues that may be uncovered during a project review. Also, because testing is performed continuously and errors are uncovered gradually throughout the project, rather than all at once upon unit integration, the impact of the errors and the resulting rework and retesting is greatly reduced. In comparison with the waterfall approach, this iterative approach reduces project risk by introducing early progress in coding, as illustrated by the following graph:


57 Rational e-Development Company, Improving Software Economics, presented in Rochester, NY by Walker Royce on 6/12/01.
The Effect of Software Requirements Analysis on Project Success and Product Quality

Vera Z. Berenbaum
Information Technology

Figure 11: Illustration of reduced project risk and shorter project schedule in an iterative development project profile as contrasted with the waterfall approach.

The aim of iterative development practices is to reduce the costs of testing and debugging by redistributing resource allocations as suggested by the Rational e-Development Company in the following table, including emphasis on management, requirements analysis, design and environment, and reducing the cost of implementation and testing:

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Waterfall Process</th>
<th>Iterative Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>Requirements</td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>Design</td>
<td>10%</td>
<td>15%</td>
</tr>
<tr>
<td>Implementation</td>
<td>30%</td>
<td>25%</td>
</tr>
<tr>
<td>Test &amp; Assessment</td>
<td>40%</td>
<td>25%</td>
</tr>
<tr>
<td>Deployment</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>Environment</td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The XP methodology stresses continuous customer involvement and satisfaction, is designed to deliver the software the customer needs when it is needed, and empowers developers to confidently respond to changing customer requirements, even late in the life cycle. The

---

methodology emphasizes teamwork between managers, customers, and developers, and simple, yet effective implementation.

XP also contributes to quality management. XP's approach blurs the line between quality planning and quality assurance. Tests are created before the code is written, while the code is written, and after the code is written. As bugs are found, new tests are added.\(^{59}\) Again, this practice is consistent with sound requirements analysis, which advocates the definition of acceptance criteria alongside each requirement statement. The acceptance criteria are then translated into system test cases. As requirements undergo changes throughout the development cycle, so do acceptance criteria and the corresponding test cases. In addition, XP requires the participation of customers and managers in the software building process, which may be limited by these stakeholders' availability. One of the limitations of XP is the relatively small size – 2-10 members – of development team XP is designed to accommodate.

4.9.2 Rapid Development and Requirements Analysis

Steve McConnell defines a rapid development project simply as "any project that needs to emphasize development speed." The basic idea behind the methodology is that effective practices speed up the project, and ineffective practices do not. Development speed is achieved by making the right choice of the following three schedule-oriented project development practices – speed-oriented practices, schedule-risk-oriented practices and visibility-oriented practices.\(^{60}\) Making the wrong choice will cause the opposite of the intended effect on the project.

McConnell indicates that each of these should be considered on a project-by-project basis. However, for any rapid development project, one of the most important pre-requisites is a clear strategy. The best possible rapid development schedule will always include the "pillars of rapid development" – classic-mistake avoidance, development fundamentals, risk management and schedule-oriented practices.

Classic-mistake avoidance is critical in rapid development. In addition to doing a number of things right, one of the most important ways to speed up development project is not doing


anything wrong. To that end, McConnell lists 36 classic delay-causing mistakes, reproduced in the table below. The mistakes that can be avoided by sound requirements analysis practices are shown in bold.

Table 16: Summary of classic mistakes that lead to slow development. Ones in bold are ones that sound requirements analysis can alleviate.

<table>
<thead>
<tr>
<th>People-Related Mistakes</th>
<th>Process-Related Mistakes</th>
<th>Product-Related Mistakes</th>
<th>Technology-Related Mistakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Weak personnel</td>
<td>15. Insufficient risk management</td>
<td>29. Feature creep</td>
<td>34. Overestimated savings from new tools or methods</td>
</tr>
<tr>
<td>5. Adding people to a late project</td>
<td>18. Abandonment of planning under pressure</td>
<td>32. Research-oriented development</td>
<td></td>
</tr>
<tr>
<td>6. Noisy, crowded offices</td>
<td>19. Wasted time during the fuzzy front end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Friction between developers and customers</td>
<td>20. Shortchanged upstream activities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Unrealistic expectations</td>
<td>21. Insufficient management controls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Lack of effective project sponsorship</td>
<td>22. Inadequate design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Lack of stakeholder buy-in</td>
<td>23. Shortchanged quality assurance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Lack of user input</td>
<td>24. Premature and overly frequent convergence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Politics placed over substance</td>
<td>25. Omitting necessary tasks for estimates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Wishful thinking</td>
<td>26. Planning to catch up later</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27. Code-like-hell programming</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.10 How Prevalent is Sound Requirements Analysis in the Software Development Industry?

Published data suggests that, with the notable exception of the U.S. Department of Defense and a few other disciplined and mature organizations, the United States software development industry does not have a good track record on taking full advantage of the benefits that sound requirements analysis offers. In introducing his book, *Effective Requirements Practices*, Ralph R. Young cites a number of examples of this problem:

In 1994, Scientific American reported [that] there exists a chronic crisis despite 50 years of "progress", suggesting that we are decades short of the mature engineering discipline required to meet the needs of our information age.62

Notable examples of this phenomenon are the Denver International Airport, California's Department of Motor Vehicles driver and vehicle registration system, and the Federal Aviation Administration's Advanced Automation System, all of which are reputed to be failures.

Capers Jones makes perhaps the most powerful statement on the effect of poor requirements analysis:

As a result [of slow progress in requirement engineering], we still see projects that skip requirements engineering entirely, or call their design a requirements specification just so it looks like they care about requirements....The progress that has occurred can best be described as "back to basics." 63

Jones describes characteristics of "back to basics" projects as follows:

<table>
<thead>
<tr>
<th>Table 17: Practices of &quot;back to basics&quot; projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintain lists of their requirements in databases (where annotating, sorting, filtering, and cross-referencing are easy)</td>
</tr>
<tr>
<td>Prototype systems before baselining requirements</td>
</tr>
<tr>
<td>Involve customers and users in the requirements process</td>
</tr>
<tr>
<td>Strive to find ways of reducing complexity and computer jargon in their requirements</td>
</tr>
<tr>
<td>Avoid approaches that claim to be panaceas</td>
</tr>
</tbody>
</table>

More evidence of the lack of requirements analysis in the software development industry comes from a recent study by the McKinsey Group, which cautions against rushing products to market.

The study attributes the recent collapse of many new economy start-ups to moving too fast to build their businesses. For most of these companies, building the business is synonymous with building Internet-based software. The study sought to determine the speed and outcome of growth of each business by studying 80 Internet companies, including business-to-consumer (B2C) companies, business-to-business (B2B) companies, and infrastructure providers. Although company growth is a result of many different factors, the study concludes that, "moving fast at the expense of developing a solid business plan and gathering the right resources rarely paid off. Speed gave an advantage to 10 percent of the companies studied, and only if certain conditions


63 Alan M. Davis, Foreward to Software Requirements Engineering, p. vii.
The Effect of Software Requirements Analysis on Project Success and Product Quality
Vera Z. Berenbaum
Information Technology

were present. When they were not, moving fast provided no discernible advantage or turned out to be costly.”

The article further states that “generally, the faster you build a business, the less time you have to study the market, test assumptions, understand competitors, and optimize resources.” In other words, moving too fast jeopardizes a company’s understanding of its business, customer and user requirements. This lack of understanding undercuts proper planning, which leads to poor quality products and services, and ultimate business failure.

Additional data on the overwhelming rate of project failure due to insufficient requirements analysis can be found in the 1995 Standish Group study. The study sought to identify three factors related to project failures – the extent of software project failures, the major factors that cause software projects to fail, and the key ingredients that can reduce project failures. The study revealed that, as of 1995, 31.1% of projects were canceled before completion, costing American companies and government agencies about $81 billion per year, and that 52.7% of projects cost 189% of their original estimates – an additional annual expense of $59 billion. The 189% cost estimate does not include the lost opportunity costs for which the industry pays additional trillions of dollars annually. Of the IT executives surveyed for this study, 48% believed the problem was getting worse. However, over 50% felt there were fewer or the same number of failures in 1995 than there were five and ten years before.

The study classified three degrees of project success. A successful project is one that was completed on-time and on-budget, with all features and functions as initially specified. A challenged project is one that was completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified. An impaired project is one that was canceled at some point during the development cycle.

---


Table 18: Cause factors leading to successful project, with requirements-related causes in bold. % of Responses

1. User Involvement 15.9%
2. Executive Management Support 13.9%
3. Clear Statement of Requirements 13.0%
4. Proper Planning 9.6%
5. Realistic Expectations 8.2%
6. Smaller Project Milestones 7.7%
7. Competent Staff 7.2%
8. Ownership 5.3%
9. Clear Vision & Objectives 2.9%
10. Hard-Working, Focused Staff 2.4%
Other 13.9%

Table 19: Cause factors leading to challenged projects, with requirements-related causes in bold. % of Responses

1. Lack of User Input 12.8%
2. Incomplete Requirements & Specifications 12.3%
3. Changing Requirements & Specifications 11.8%
4. Lack of Executive Support 7.5%
5. Technology Incompetence 7.0%
6. Lack of Resources 6.4%
7. Unrealistic Expectations 5.9%
8. Unclear Objectives 5.3%
9. Unrealistic Time Frames 4.3%
10. New Technology 3.7%
Other 23.0%

Table 20: Cause factors leading to impaired projects, with requirements-related causes in bold. % of Responses

1. Incomplete Requirements 13.1%
2. Lack of User Involvement 12.4%
3. Lack of Resources 10.6%
4. Unrealistic Expectations 9.9%
5. Lack of Executive Support 9.3%
6. Changing Requirements & Specifications 8.7%
7. Lack of Planning 8.1%
8. Didn't Need It Any Longer 7.5%
9. Lack of IT Management 6.2%
10. Technology Illiteracy 4.3%
Other 9.9%
Requirements analysis and user involvement, the latter being a fundamental part of sound requirements analysis, appear at or near the top of each list of the factors cited. The study clearly demonstrates the great impact of sound requirements analysis on the success of software development projects. More importantly, it underscores the overwhelmingly negative effect of incomplete requirements and lack of user involvement in the 31.1% of the software projects that failed in 1995.

A 1992 study showed that 63% of large software significantly exceeded their cost estimates. That study also found that the following four reasons were most frequently cited for this problem – frequent requests for changes by users; overlooked user tasks; users' lack of understanding of their own requirements; and insufficient user-analyst communication and understanding.66 Again, the four reasons point to the fact that inadequate user requirements analysis and user engagement in the development process leads to overly expensive software development projects. A 2001 article echoes these findings with the following six reasons for IT project failure – lack of executive sponsorship; lack of early stakeholder input; poorly defined or changing specifications; unrealistic expectations; uncooperative business partners; and poor or dishonest communication.67

5. Impact of Requirements Analysis on Product Quality

5.1 Quality Management Depends on Sound Requirements

Quality management is a complex job, involving the coordination of many processes and techniques. Some of these are mutually dependent, and required for every project. Others are invoked on an as needed basis. The Project Management Book of Knowledge (PMBOK) provides a detailed description of project quality management.68 At a high level, project quality management consists of quality planning, quality assurance and quality control.

Quality planning is conducted early in the project, concurrently with, and immediately following the definition/analysis phase. Quality planning is heavily dependent on functional requirements.


Each time there is a requirements change, quality planning must be revisited to reflect the changes in the project schedule and account for them in quality assurance.

Although the activities and resources related to quality assurance are planned in the early stages of the project, QA’s main contribution comes at the end of the execution phase, when the greatest number of resources is engaged. Any defects uncovered at this time require rework, cost and time. While both quality planning and quality assurance strive to improve quality by minimizing the number of defects, quality planning focuses on preventing errors, whereas quality assurance focuses on fixing errors through rework. It stands to reason that the time, effort and cost invested in quality planning would reduce time, effort and cost invested in quality assurance. Although the costs of QP and QA are inverses, the overall cost of QP is much smaller than that of QA, as shown in the graph below.

![Figure 12: Differences in costs of quality planning and quality assurance.](image)

There are three main reasons why investing in quality planning is good for the project. One difference is that quality planning costs less overall because its greatest contribution comes at the beginning of the project when the fewest number of resources is engaged, and any resulting rework is minimal. A similar defect discovered by QA would entail rework at a much greater cost in terms of resources, time and team morale. The second reason is that investing in quality planning reduces, which is already cheaper than quality assurance, has the effect of bringing down the cost of quality assurance. Any defect that is prevented early in the project, will not be
discovered by QA later in the project, and will not cost extra to fix. The third difference is the timing of the defects. When uncovered at the end of the project, QA defects extend the project timeline more, putting the delivery date at greater risk.

The quality assurance phase of a project is generally much more resource-intensive and costly than the quality planning phase. The bulk of quality planning can be accomplished early in the project, with minimal staffing and cost. Therefore, the initial investment of time and resources in quality planning is less costly per error than the corresponding investment in quality assurance.

5.1.1 Quality Planning

Quality planning is overwhelmingly dependent on adequate definition and understanding of the project scope, which is derived directly from requirements. The PMBOK describes quality planning as the process of identifying quality standards that are relevant to the project and determining how to satisfy them. For most projects, quality planning consists of a number of planning and facilitating processes, as shown in the table below.\(^69\) The core processes must be conducted in the specified sequence, whereas the facilitating processes are invoked as needed.

<table>
<thead>
<tr>
<th>Table 21: Quality Planning Core Processes</th>
<th>Table 22: Quality Planning Facilitating Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Scope planning</td>
<td>• Quality planning(^70)</td>
</tr>
<tr>
<td>2. Scope definition</td>
<td>• Organizational planning</td>
</tr>
<tr>
<td>3. Activity definition</td>
<td>• Staff acquisition</td>
</tr>
<tr>
<td>4. Activity sequencing</td>
<td>• Communications planning</td>
</tr>
<tr>
<td>5. Activity duration estimating</td>
<td>• Risk identification</td>
</tr>
<tr>
<td>6. Schedule development</td>
<td>• Qualitative risk analysis</td>
</tr>
<tr>
<td>7. Risk management planning</td>
<td>• Quantitative risk analysis</td>
</tr>
<tr>
<td>8. Resource planning</td>
<td>• Risk response planning</td>
</tr>
<tr>
<td>9. Cost estimation</td>
<td>• Procurement planning solicitation planning</td>
</tr>
<tr>
<td>10. Cost budgeting project plan development</td>
<td></td>
</tr>
</tbody>
</table>

The first two core processes – scope planning and scope definition – are impossible to accomplish without some understanding of business, user and system functional requirements. Since the remaining eight core processes are functionally dependent on scope planning and scope definition, it follows that they are also highly dependent on well-defined and understood requirements.


\(^70\) Quality Planning is an iterative process that must be revisited as dictated by the CCB and other changes to the project plan throughout the execution of the software project.
The facilitating processes may be performed intermittently as needed throughout the project, but
are by no means optional. The need for facilitating processes is usually determined by changes
in project scope, arising from newly identified needs or from requirements that were missed
during the analysis phase of the project. Ideally, the scope changes are identified through the
change control process. The quality planning and organization planning facilitating processes re-
introduce planning into the project to mitigate newly identified risks. Addressing new scope
issues without planning for them will have the effect of over-extending the resources and putting
the project behind schedule.

5.1.2 Quality Assurance

Quality assurance is universally viewed as the number one test for gauging project success in
terms of quality. If the product passes an acceptable number of quality tests, the project quality
is deemed successful. However, the greatest contribution of quality testing is the assurance that
the project has met its functional requirements. If the functional requirements reflect usability
considerations, then passing QA tests based on these requirements measures the success of both
the project and the product. Otherwise, the test measures the success of the project only. QA
tests that are not based on requirement acceptance criteria, neither demonstrates that the product
was built to specification, nor that the project has met its quality targets. If requirements don’t
reflect change control results, testing against them falls into the latter category. Therefore, a
requirements document that is not updated throughout design and implementation has little or no
value in QA testing. Testing against outdated or obsolete requirements may result in undue
project failure or undue success, regardless of the validity of either conclusion. Such tests serve
only to re-define project success.

The best practices in quality depend on tracking functional requirements throughout the duration
of the project life cycle. This is accomplished by making certain that requirements are updated
each time a change is warranted and approved by a critical mass of stakeholders, and that quality
tests are written against the most recent set of requirements.

5.2 Quality Acceptance Criteria

Requirements analysis is an integral part of quality management because requirements set the
criteria by which quality tests determine whether the required functionality has been delivered.
Each project has requirements. Requirements may be clearly defined and documented, or merely
implied by the customer and inferred by the project team. They may be expressed verbally, and they may undergo a series of changes throughout the project. In order for everyone to agree that development is complete, all stakeholders must have a common understanding that whatever product was set out to be built, has been completed.

If the project followed a waterfall development approach, and requirements were not permitted to change significantly throughout implementation, it is likely that customer needs have shifted, and they will not be happy with the project. In this scenario, the original set of documented requirements is the project team’s only evidence that the project has in fact met agreed-upon quality targets, and a well-managed requirements document is very valuable.

If the project followed an iterative development approach with continuous customer involvement, chances are most customers will be pleased with the outcome. However, if there is any disagreement between camps of customers, or if there are any deviations from the original set of requirements that did not have buy in by all stakeholders, the quality of the product may still be questioned. A well-maintained requirements document that reflects all the scope changes incurred during iterative product reviews will serve as evidence that the moving quality targets have indeed been met.

5.3 Product Success

5.3.1 Definition of a Successful Product

If project success is determined by whether or not quality targets are met within budget and on time, product success is determined by whether or not customer expectations are met. Functional requirements that do not meet customer expectations may yield a successful project that results in an unsuccessful product. Requirements analysis practices are the foremost way to bridge the dissonance between project success and product success.

A product is completely successful if it has met all the criteria for its success, as defined by the functional requirements. Once a product is released, there is a number of valid ways to measure the success of a commercial software product, including, but not limited to usability, short-term sales, long-term sales, subscription renewals and customer feedback.
5.3.2 Usability

Software product usability is arguably the most important factor in defining the long-term business success or failure of a software product. Persuading a consumer to invest in a software package without having tried it is easier than persuading him to purchase an upgrade once he has been disappointed by the product. The easiest way to persuade a customer to purchase an upgrade is with the argument that the upgrade is even better than its predecessor, which the customer has tried and liked.

Usability can be measured by testing end-users’ interaction with the system, conducting customer surveys, studying customer feedback to customer support, and examining the types of questions customers ask about the product.

5.3.3 Short-term sales

One measure of software product success is the immediate response it sees on the market. If the software product is the first of its kind to market, immediate short-term success will be based more on its perceived value and demand than on any other product success variable. In building a first-of-a-kind product, it makes sense to focus the software project on the timeline, rather than any of the other project management constraint. However, in the absence of sound user interaction design, the success of even a first-of-a-kind product is likely to be short-lived. Competitors will quickly bring to market similar products and capitalize on their improvements over the current standard.

5.3.4 Long-term sales

A consumer will tend to purchase good software products repeatedly with subsequent releases or upgrades. Although long-term sales are largely influenced by a product’s usability, usability is not the only factor that determines how well a product will do on the market long-term. Marketing, sales, competition, perceived demand, economic conditions, and a number of other factors will have an effect on long-term sales. Therefore, long-term sales can be taken as a measurement of product success that is separate from usability.

5.3.5 Subscription renewals

Subscription renewals are similar to long-term sales, but they usually pertain to web-based software products with access restricted only to paying subscribers. Examples include Web sites
used for publications, consulting services, research, reference, and distribution of pornography. If a site is confusing, distracting, or in any way unwelcoming, the customer is less likely to return to it and therefore will not renew his subscription the next time it is up for renewal.

5.3.6 Customer feedback

Customer input can be sought in many ways, for example through customer surveys, informal queries, and solicited or unsolicited feedback to customer support staff. For example, if the first day the product is on the market, Customer Support gets a flurry of calls about users reporting a bug that is preventing them from accessing the product, this can be taken as bad news about the quality of the product. In addition, types of questions presented to Customer Support may be analyzed for common themes and patterns. This information can then be used to judge the quality of the software product.

5.4 Ways In Which a Product Can Be Unsuccessful

There are four primary stakeholders that are involved in defining product success. These stakeholders include corporate executives, software developers, human factors specialists/customers, and project managers.

A corporate executive will define an unsuccessful product as one that does not make a profit. A human factors specialist will define an unsuccessful product as one that is not usable. This definition allows for the possibility that a well-selling product that might still be considered unsuccessful. A software developer will define a successful product as one that meets all its requirements or as an example of particularly good code. Finally, a project manager will tend to mis-define a product as successful if it is the outcome of a successful project – that is, if it is delivered on time, on budget, and within stated scope. As already discussed, a successful project does not always result in a successful product.

5.5 System Acceptability

Jakob Nielsen offers another useful way of evaluating product success is in terms of system acceptability. Nielsen's model of the attributes of system acceptability\(^\text{71}\) consists of social acceptability and practical acceptability, as shown in the figure below:

---

Nielsen describes social acceptability as a reaction to what the system does, rather than how well it performs that function. For example, an x-ray vision machine may perform full body searches very well, but most United States citizens would probably find the violation of civil rights socially unacceptable. From that point of view, the system would be unacceptable overall, even if it is perfectly functional and fully meets its users’ requirements and expectations. Although outside the scope of this paper, social acceptability is an important consideration in determining product success or failure.

5.6 Creative Ways to Re-Define Product Success

The most common way to mis-define product success is by confusing it with project success. If the project has delivered all the functionality it had been tasked with, and has met its quality criteria, it may also have delivered a successful product. For the product to be successful, the original scope had to have been defined and diligently analyzed with participation from all the appropriate experts, such as the customers, human factors specialists, market analysts, and business analysts. Furthermore, all these experts had to have been correct in their assumptions about customer needs, the market and the business, and their specifications had to not have been diluted by scope negotiations, resource constraints, and other project management considerations.

Whether or not a software product is successful cannot be truly known until it has been released, has seen initial sales success, has been used, re-used and appreciated, and purchased and repurchased by new and old customers. Until all this data is available as evidence, a product cannot be accurately labeled as “successful.”
5.7 Definition of Product Management

As with most disciplines, there is a number of ways to define “product management”. The PMBOK offers a good starting point for defining product management, by stating that product management focuses on the features and functions that characterize a product or a service. A product scope statement would describe the desired outcome of a project in terms of the features and functions of the resulting product. The success of a product management would then be measured in terms of sales, usability of the product, and customer satisfaction.

In contrast, a project is defined as a temporary endeavor undertaken to create a unique product, service or result. Therefore, project management focuses on the work that must be completed in order to deliver a product with the maximum number of specified features and functions. A typical project plan will be based on product requirements, but will balance these requirements against issues of cost, timelines and resources as well. The success of a project is measured in terms of budget, deadlines, meeting product requirements, and employee satisfaction.

<table>
<thead>
<tr>
<th>Table 23: KLCI list of “Eight Key Questions” that span the spectrum of Product/Business Planning needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What is the solution (market and product requirements)?</td>
</tr>
<tr>
<td>2. Who is the customer (target markets segments)?</td>
</tr>
<tr>
<td>3. Why will the customers buy it (key purchase criteria)?</td>
</tr>
<tr>
<td>4. What is the value that my company brings to the solution (value proposition)?</td>
</tr>
<tr>
<td>5. How will the product be sold (direct/indirect channel strategies)?</td>
</tr>
<tr>
<td>6. How will my company win versus competitors (competitive differentiators)?</td>
</tr>
<tr>
<td>7. How big is the financial opportunity (return on investment)?</td>
</tr>
<tr>
<td>8. How much risk is in the program (opportunity costs and other risks)?</td>
</tr>
</tbody>
</table>

5.8 Effect of Requirements Analysis on Product Success

If done adequately, functional requirements analysis establishes criteria by which product success will be judged upon completion of the project. Product success is further tested by

---


74 Software Marketing and Product Management Overview, KLCI Research Group. The KLCI Research Group provides consulting services to software development organizations on a number of topics, one of which they call Product/Business Planning. http://www.kliciresearch.com/software/overviewswpm.html, last viewed on 4/18/01.
customers and the market after the product’s release, but the quality of the functional analysis is tested even in the latter situation, long after the project is complete. Unfortunately, it is not only possible, but even common, for functional requirements to lead to successful project completion, and still miss the points that would make the product successful after project completion and the release of the product. This is evidenced by the numerous software products that may have been released on time, on budget, and met some sort of project quality criteria, but that are unusable or do not sell. This is a direct result of poor requirements analysis – one that was done with inadequate user engagement, usability considerations, or poor business market analysis.

All too often software functional specifications are system-driven and arrived at in a vacuum. The forces that influence product success are many. They include market research, customer feedback, human factors analysis, budget and resource considerations, as well as priorities for possible funding of competing projects, and, last but not least, organizational politics. Ideally, functional requirements definition and analysis should be performed concurrently with business analysis, human factors analysis and market analysis, and should begin at the time of project selection.

A project may be selected for a number of reasons, not all of which may be conducive to product success. For example, the primary motivation for a product might be to keep up with competitors who may be selling a similar feature or product, successfully or otherwise. The management may believe that such a feature is essential and desired by customers. Furthermore, the management may be of the opinion that time to market is the most critical issue to address because of the perception of “missed opportunity”. However, rushing into a project to create such a product would be misguided unless it is accompanied by empirical evidence that customers would actually appreciate and/or buy the product and, most importantly, that they would use it and recommend it to their friends. Ideally, this study of the product’s viability would come before the definition of functional requirements. If it does not, it is part of the requirements analyst’s job to ask the right questions to bring out this information. Writing functional specifications without an understanding of the business drivers and human factors behind the product may result in a successful project that yields an unsuccessful product.

By the time requirements are agreed upon and documented, all the business drivers and human factors considerations should be established and agreed upon as well. At the very least, they should be clearly documented in hopes that the information will help management and stakeholders come to a more informed decision. It may be that producing an unusable product or
feature still makes business sense if short-term sales expectations are high enough and the long-term cost is offset in ways that make business sense. The requirements assumptions should then be validated through prototyping, preferably before a project is planned and executed.

5.9 Recommendations to the Product Manager

The most important advice to the product manager is not to limit the duration of the definition/analysis phase of the project. Requirements analysis performed under great pressure will either fail to identify the project risks, or will fail to resolve them. The cost of an adequate definition/analysis phase is more than realized in the later phases of the project when it causes fewer unforeseen implementation problems and shortens the duration of the quality assurance phase.

As demonstrated throughout this paper, the most effective way to produce a quality product is to elicit and validate the correct set of business and user requirements, and then meet them. Eliciting and validating requirements takes time. However, the chief problem with the very important analysis and prototyping techniques is that, since they contribute to the product deliverable, but do not constitute the deliverable in and of themselves, they tend to get undervalued by project managers, especially in time-challenged projects.

In projects where deadlines must be negotiated between project managers and requirements analysts, the presence of data or process diagrams in the requirements document tends to be viewed as optional, and the argument that it is the basis for the completeness and accuracy of each requirement statement tends to be dismissed. Also dismissed is another important function of visual representations of the requirements is fostering a common understanding among stakeholders a la “a picture is worth a thousand words.”

One of the most important contributions to product quality that a project manager can make is to insist on thorough definition, analysis and validation of requirements. With this, the extensive testing that must follow any quality-centered project will be more fruitful and result in less time-consuming rework. Most importantly, the product will be in line with users’ and customers’ expectations.

Keep customer expectations in line with the resulting product’s functionality, customers’ needs and interests must be monitored and fine-tuned throughout the product development cycle. As noted elsewhere in this paper, developers are apt to make inappropriate assumptions or to alter
requirements throughout the development process. The requirements analyst must be involved and aware enough to influence these changes before they take the development effort in the wrong direction. If this divergence from functional requirements is left unchecked, it can lead to project and product failure.

5.10 Assessing the Value of Sound Requirements Analysis

Of all the project phases and tasks that go into building a software product, the value of the requirements analysis effort is perhaps the most difficult to measure. The value of good planning and scope control is intuitively obvious to anyone who has ever worked on a project. It is easy to conceive the fact that without adequate planning and scope control, the project would have failed, would have cost more, or would have resulted in a more defective deliverable. It is also not difficult to accept the fact that preventing errors is far cheaper than finding and correcting them after the product is built. Yet, the person responsible for finding errors – the quality tester – is in a much better position to quantify and demonstrate her contribution than the person responsible for preventing errors – the quality planner, usually the requirements analyst.

To demonstrate her contribution, a quality tester can simply point to the number and severity of "bugs" that were found during product testing. The requirements analyst may have prevented twice as many bugs that might have been even more severe, simply by thinking through all aspects of the software product ahead of time. Unfortunately, he does not have the convenience of having those errors neatly captured in a spreadsheet, and the drama involved in fixing them at the last minute and rescuing the project. In a sense, the requirements analyst's job is to reduce the amount of drama and unpredictability involved in disorganized software development efforts. This predicament leads to difficulty in justifying an adequate requirements gathering effort for future projects, thus affording more opportunity for after-the-fact errors for quality testers to find.

How then, does a requirements analyst justify her existence to a manager who trusts only quantitative data?

Managers who with a poor understanding of the requirement analyst's contribution, attempt to quantify the benefit of the requirements analyst by benchmarking the initial set of requirements for a project, and then tracking how many change throughout the project. In this scenario, the more requirements change throughout the project, the worse the requirements analyst's
performance. For example, an analyst may write down 15 requirements for a software project, then get together with the project team to review these requirements. The project team may identify 3 more, and alter 5 of the original 15 requirements. This would mean that the analyst was “right” in identifying 10 of the 18 “correct” requirements, and would earn a rating of about 60% before the requirements are even agreed to by all stakeholders. Furthermore, if these 18 requirements change further during implementation, the requirements analyst’s rating would be lowered even further, below 60%.

This system misrepresents the contribution of the requirements analyst as getting the requirements “right” the first time, rather than eliciting them from stakeholders, building consensus, preventing errors and lowering overall project costs. Furthermore, this system has a built-in disincentive for the requirements analyst to find and correct inconsistencies and inaccuracies in the requirements document, as doing so would result in a lower performance rating.

A better system would be one where a relatively large sample of projects is compared phase for phase, both quantitatively as well as qualitatively. Perhaps an organization has seen a couple of software projects that included a particularly generous amount of time for requirements analysis, and a couple of similar projects where the analysis phase was abbreviated. What differences can then be observed in the quality testing phases of those projects? Presumably, the more detailed the requirements in the project’s quality planning, the fewer bugs should be found in the quality assurance phase. Do that quantitative comparisons bare this out?

If the organization does not have enough of its own projects to study, perhaps a better idea is to sample colleagues in similar positions in other companies. Here are some sample questions that would be helpful to ask.

6. Impact of Requirements on Project Success

6.1 Emerging Trends in Project Management are Not Successful

Project management is a mature, well-established, time-tested discipline. Any project, formal or otherwise, goes through some form of definition/analysis, planning, execution, testing and close down phases. This description applies both to waterfall and iterative approaches to project management. Structurally, a software project does not deviate much from any other kind of

---

75 This example is taken from the author’s own professional experience. No literature sources were
project. However, businesses in the emerging “new economy” have shown great reluctance to adhere to the time-proven principles. In the interests of shortening projects and releasing products to market more quickly, they tend to take less time to analyze customer and market needs to establish a direction and structure for growth. Already discussed are the results of the recent McKinsey study that lamented the poor success these businesses have seen.\textsuperscript{76}

Other studies show similar trends emerging in project management. In his summary of a study on the differences between traditional and emerging practices in software development projects,\textsuperscript{77} Don O’Neill indicates that traditional software projects tended to be supplier-driven. Traditional project management values included commitment management, repeatability in project practice, predictability of cost and schedule performance, and orderliness in operations. Traditionally the valued management track practices were requirements determination and management, estimation and planning, tracking and oversight, configuration management, and quality assurance. In addition, the valued engineering track practices were based on the sequential waterfall life cycle model featuring requirements, specification, design, code, and test in developing software systems from the ground up.

As for newly emerging software projects, O’Neill’s study indicates that they tend to be market driven. Today, the valued management track practices revolve around systems integration featuring a commercial off the shelf product or an enterprise legacy system as a product baseline and an evolutionary development process with a never ending informal requirements process and a successive iteration of incremental releases. Furthermore, the valued engineering track practices are architecture views followed by repetitions of code and upload. New values include early delivery of product to customers, innovation in technical approach, boldness in management approach, process of experimentation, decriminalization of defects, taking calculated risks, civility, push back, cooperation, collaboration, and competition. The current


and emerging state is characterized by a set of preferred organizational and project behaviors that are market driven and permit the software project to find its own patterns for success... or failure.

<table>
<thead>
<tr>
<th>Table 24: O’Neill’s Characteristics of “old style” project management behaviors</th>
<th>Table 25: O’Neill’s Characteristics of “newly emerging” project management behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Requirements known fully at the beginning</td>
<td>1. Requirements known at the end</td>
</tr>
<tr>
<td>2. Top down design and development</td>
<td>2. Middle out development</td>
</tr>
<tr>
<td>3. Big bang delivery</td>
<td>3. Incremental, iterative, experimental</td>
</tr>
<tr>
<td>4. Hard and fast plans</td>
<td>4. Easily changed</td>
</tr>
<tr>
<td>5. Cost plus, full scope contracts</td>
<td>5. Fixed price, task order contracts</td>
</tr>
<tr>
<td>6. Schedules in years</td>
<td>6. Schedules in months</td>
</tr>
<tr>
<td>7. New development</td>
<td>7. Reuse and off the shelf</td>
</tr>
<tr>
<td>8. Quality priority</td>
<td>8. Time to market</td>
</tr>
<tr>
<td>12. Risk Adverse</td>
<td>12. Calculated risk</td>
</tr>
<tr>
<td>13. Individuals</td>
<td>13. Empowered teams and push back</td>
</tr>
<tr>
<td>14. Value superior power</td>
<td>14. Value superior knowledge</td>
</tr>
<tr>
<td>15. Intramural cooperation</td>
<td>15. Intermural collaboration</td>
</tr>
</tbody>
</table>

O’Neill observes the following consequences of the new values and behaviors:

1. Earlier preferred practices and behaviors applied today lead to poor outcomes in claiming market share with the principal shortfall being time to market.
2. Today’s preferred practices and behaviors lead to poor outcomes in the trustworthiness of software systems with the principal shortfalls being the reliability of their operation, the accuracy and protection of the data they produce, and the cost of sustaining operations.

To summarize O’Neill’s observations, the major difference between “traditional” software projects and the emerging software projects is the focus on time to market at a cost of sacrificed planning and poor quality. Combined with the McKinsey findings that rushing projects does not result in shorter projects, one wonders why an organization would even try to carelessly shorten a project. Nevertheless, advocates of rapid development attest that there are ways of “intelligently” shortening projects. Upon close examination, their primary audience is development shops that have abandoned traditional methods, and their advice advocates a return to the traditional project management approach by focusing on defining the problem, planning and controlling the project and reducing project risk. The results are shorter projects and better quality results.
6.2 Why Shorten Projects?

Shortening project duration appears necessary when the market value of the product under development is perceived to be time-sensitive. The following diagram illustrates the difference between traditional products and one that have strong schedule constraints\(^\text{78}\):

Figure 14: Value of typical vs. time-constrained products over time

Another common reason for shortening project durations is to live up to the overly optimistic predictions that tend to be made to project sponsors by development managers in securing funding for the projects. Boehm attributes this tendency to underestimate software size, and therefore project schedules, to the fact that people are basically optimistic and desire to please, tend to have incomplete recall of previous experience, and are generally not familiar with the entire software job.\(^\text{79}\)

There are some legitimate business advantages to be the first to market with a new product. In these situations, it often makes business sense to take a risk because of the huge market advantages of having a first-of-its-kind product. However, a great number of companies gamble at a high risk even if their product is not the first to market, but is intended merely to keep up with or slightly improve upon competition. In such situations, the company is better off


investing time and resources in the product's quality, so that when it is released, it truly is an improvement on its competition.

6.3 How to Shorten Projects with Minimal Risk

Regardless of business sense, shortening development projects is for now a way of life. Fortunately, there are also plenty of effective ways to shorten software projects. In terms of project costs, if the project duration is to be shortened cost-effectively, it is best done in one of the more expensive phases of the project—during implementation and quality testing, as illustrated by the following diagram of activity/cost levels on a typical project. ⁸⁰

Figure 15: Activity and cost on a project are lowest during the definition/analysis and planning phases. Any errors caught during these phases will be significantly easier and cheaper to fix than they would be if discovered later in the project and required rework and retesting. Slightly modified from Haynes.

One of the most effective ways to shorten a project is by reducing the number of errors discovered in testing. Investing in definition/analysis and planning is not only relatively cheap, but it also has the effect of reducing QA time and shortening project duration, minimizing project risks, and improving product quality. Reduction of the defect rate is critical because a high

defect rate tends to greatly increase development time, thereby increasing project costs and the chance of schedule slips. This idea is demonstrated in the following diagram:

Figure 16: Relationship between defect rate and development time. Projects that achieve the lowest defect rates also tend to achieve the shortest schedules.

One of the most effective ways to reduce defects is to prevent them through effective quality planning. Shortening the definition/analysis phase tends to result in more errors uncovered in implementation and testing. The subsequent rework will result in lengthening project duration rather than shortening it.

The deliverable from the definition/analysis phase – the documented requirements specification – is the basis for planning the rest of the project in the first place. In terms of project cost, allowing some slippage in this phase, or better yet, not rushing it, is more than made up for by preventing errors that would otherwise be discovered later in the project. Slippage during the definition/analysis phase is usually caused by discovery of problems that would have otherwise been identified in a later, more expensive phase of the project. A quality gate assessment here is much cheaper both in terms of cost and in terms of project time. Any re-planning and re-implementation that takes place later in the project is much more costly, more difficult to control,

---

and will likely cause rework and even greater schedule slippages than those that would be required during the definition/analysis phase.

Mass and Berkson\textsuperscript{82} give an excellent illustration of the greater value of quality planning over quality assurance, to project success. They state

> If problems are not detected and corrected at the earliest upstream gates, downstream quality gates end up pulling resources from one project to another, which leads to multiple schedule delinquencies, poorer design stability, and quality shortcuts prompted by intensified schedule pressures. It seems likely that companies utilizing multiple quality gates across design and manufacturing may be unintentionally reducing their profits and quality. The challenge for management is to design quality gates that are effective, efficient, and positioned as early as possible in the development process.

Mass and Berkson describe the use of “phase gates” by a number of industries, to check product quality during the production cycle. One of this is a quality gate, which is defined as “a ‘hard’ screening and rework point beyond which products cannot pass until they are upgraded to at least the gating quality level.” Mass and Berkson further point out that “…a quality gate upstream in concept development can actually help boost profits by over 12 percent. However, downstream quality gates add progressively less value and can eventually destroy value... [while] a quality gate positioned behind the prototype stage or in integrity testing is worse than no quality gate at all, reducing total profits by 4 to 5 percent.

**Figure 17:** Mass’s and Berkson’s conception of the profit impact of quality gates based on their timing during the project. The earlier in the project the quality gate takes place, the greater the pretax profits.

![Profit Impact of early and late quality gates](image)

In addition, Mass and Berkson cite evidence that increasing emphasis on quality assurance testing increases project cost and extends the timeline due to the rework inspired by the problems uncovered by quality tests. This is an example of a quality gate taking place too late in the production cycle. The fact that errors were uncovered before product release is good news,

However, adequate quality planning (based on solid requirements analysis) would have the same effect on the product without the additional cost and schedule overruns.

Although no amount of time during definition/analysis will guarantee that the project schedule will not slip later on, the identification of potential project risks is much more welcome during the analysis phase than during implementation or testing. Furthermore, the nature and the purpose of requirements analysis are to generate more questions than answers. Each unanswered question constitutes a project risk. At the very least, the uncertainties must be documented so that the project manager may take them into account in planning and scheduling. At best, these uncertainties should be resolved prior to planning and scheduling.

### 6.4 The Definition of a Successful Software Project

A software development project can be called successful is if it has met its quality targets, schedule and projected cost. This is a standard definition. Ed Yourdon, for example, defines a successful systems development project as one that is delivered on schedule, relatively close to budget, and with high levels of quality and reliability. However, a project can still be called successful even if is not delivered on the original schedule, close to the original budget, or to the original scope. If the decisions to depart from the original project plan were made consciously, the project scope was updated to reflect the changes, and resulted in all the necessary fortifications to the budget and the schedule, and the new benchmarks were met, then the project can still be called successful.

A project that at the outset is understood to be a “best effort” project, rather then one with a hard deadline, budget and scope cannot be considered a failure. A projects whose scope and benchmarks are defined or re-defined after completion can be “sold” as successful. In reality, the success of such a project is indeterminate.

---

6.5 Creative Ways to Redefine Project Success

Kulik and Samuelson\textsuperscript{84} assert that success of high tech projects (e-projects) is defined differently from that of traditional projects because e-projects are more time-constrained than they are budget- or resource-constrained. Therefore, e-projects pose unique challenges to project managers. Kulik and Samuelson quote a study by McKinsey and Company that "found that high-tech projects completed on time but up to 50 percent over budget were nearly 25 percent more profitable than projects completed six months late but on budget," and assert that these percentages are "amplified" in "today's e-business environment". The implication is that these projects can be considered successful.

Kulik and Samuelson define an e-project as "any project that involves creating or changing source code that is deployed on the Internet ... [including] a range of projects from new content deployment in HTML to applet enhancements in Java and ActiveX. The key differentiator between e-projects and "traditional [software development] projects" are release frequency and software size. Where "traditional [software development] projects" are six months or longer in duration, a typical e-project would span a period of "days to a few months." Additional key attributes of e-projects include their iterative nature ("Incremental new features can be introduced – and bugs fixed – in real time.") and the fact that speed is of the utmost importance, at the expense of "scope, cost and/or quality to accelerate the speed of project completion."

Finally, three "classes" of e-projects are described as: new construction (usually a "traditional project"), remodeling (usually an e-project), and maintenance (usually an e-project). The authors summarize the differences between traditional and e-projects as follows:

<table>
<thead>
<tr>
<th>Project Activity</th>
<th>Traditional Projects</th>
<th>e-Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Gathering</td>
<td>Rigorous</td>
<td>Limited</td>
</tr>
<tr>
<td>Technical Specifications</td>
<td>Robust</td>
<td>Descriptive overview</td>
</tr>
<tr>
<td>Project Duration</td>
<td>Measured in years</td>
<td>Measured in days, weeks or months</td>
</tr>
<tr>
<td>Testing and QA</td>
<td>Focused on achieving quality targets</td>
<td>Focused on risk control</td>
</tr>
<tr>
<td>Risk Management</td>
<td>Explicit</td>
<td>Inherent</td>
</tr>
<tr>
<td>Half-life of Deliverables</td>
<td>18 months or longer</td>
<td>3 to 6 months or shorter</td>
</tr>
<tr>
<td>Release Process</td>
<td>Rigorous</td>
<td>Expedited</td>
</tr>
<tr>
<td>Post-release Customer</td>
<td>Requires proactive effort</td>
<td>Automatically obtained from user interaction</td>
</tr>
<tr>
<td>Feedback</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textsuperscript{84} Kulik, Peter and Robert Samuelson, "e-Project Management for the New e-Reality", \textit{PM Network: The Professional Magazine of the Project Management Institute}, March 2001
The rightmost column implies that e-projects are inherently different from traditional projects in the areas of quality planning, quality assurance and usability analysis. No differences are noted in design and development phases. The authors appear to be arguing that shortening the non-design and non-development phases of the project will tend to shorten the duration of the entire project and get the product to market faster. In fact, they are demonstrating why e-projects are less successful than traditional projects. Kulik and Samuelson contradict the advice given in *Rapid Development* on how to shorten software development projects successfully, and is not supported by lightweight methodologies, which advocate continuous requirements analysis, user involvement and testing. It would appear that using Kulik’s and Samuelson’s approach would result in increased project delays and failures.

### 6.6 How Successful Are Software Projects?

There appears to be a consensus in the IT industry that the vast majority of software projects are not successful. The cover page of a Project Leadership Conference invitation to take place in June 2001, states, “Of all the IT projects in the U.S. – 72% will fail or be challenged.”

Steve McConnell extrapolates the following data from his research:

> “About two million people are working on about 300,000 software projects in the United States [circa 1997]. Between one third and two thirds of these projects will exceed their schedule and budget targets before they are delivered. Of the most expensive software projects, about half will eventually be canceled for being out of control. Many more are canceled in subtle ways – they are left to wither on the vine, or their sponsors simply declare victory and leave the battlefield without any new software to show for their trouble.”

McConnell further notes that “software projects succeed or fail based on how carefully they are planned and how deliberately they are executed. The vast majority of software projects can be run in a deterministic way that virtually assures success.” Furthermore, we have already

---


discussed is the McKinsey report that only 10% of internet businesses derive success from time-driven projects.88

6.7 Effect of Requirements Analysis on Project Success

The factors determining project success are cost, time and quality. None of these can be baselined without a clear understanding of the project’s scope, which in turn is based on the functional requirements. Without requirements definition, none of the three key factors of project management can be baselined or accurately estimated. Each time there is a possible change to project scope, it must be validated against existing requirements to be considered for inclusion into the project.

By far the greatest contribution of requirements analysis to a project is in the area of quality. Requirements analysis is integral in quality planning and management. Quality planning is balanced against quality assurance, which is one of the most expensive steps in the project. Passing quality assurance tests is a powerful way for a project manager to demonstrate project success as it relates to quality. Quality assurance tests are generally written against functional requirements. A project cannot be called successful unless it has met criteria of quality as defined by the functional requirements. And a product cannot be called a quality product unless it has met quality targets, as specified by the requirements acceptance criteria.

6.8 Recommendations to the Project Manager

6.8.1 Do not limit the requirements analysis

A requirements analysis effort that takes longer than expected because new issues are uncovered and new questions are asked, means that implementation and testing phases will have fewer surprises. Do not attempt to abbreviate this phase out of fear that it prolongs the project. The opposite is likely to be true. The project has already started. All you’ve done is front-load the troubleshooting. If there are reasons to reschedule or not to take on the project, it’s best to find them out now rather than later.

6.8.2 Do not undertake a project without understanding its full scope and the organization’s capabilities.

An organization should never find itself in a situation where it hits the limits of its abilities before the requirements are well-understood, or after the analysis phase of a project. The key is to have an assessment of an organization’s abilities at the end of the analysis phase, but before implementation. The following diagram illustrates this point:

Figure 18: Once requirements are baselined, no project should be planned or implemented unless the project manager has assessed that the organization has adequate resources to meet the requirements.

6.8.3 Resist shortening projects at the expense of quality and usability

The only time it makes sense to rush a product to market is if it is the first of its kind. Even so, the product needs to be useful and usable enough to take attract customers in the long-term. Otherwise, your company will lose the number one status to the first competitor with a moderate improvement on your product. If the aim of the product is to keep up with competition, it almost never makes sense to jeopardize quality and usability in the interests of speed.

If shortening the project is necessary, one key activity not to cut short is the requirements definition/analysis phase. In fact, it is advisable not to include it on the project clock if at all possible. The project manager should take every advantage of ill-defined project scope as an excuse to delay planning the project. The more of the project is defined and planned up front, the more predictable and the less risky the project will be, and the more likely it is to be successful. It is better to give the requirements analyst time to predict and prevent errors during the early and cheap phase of the project life cycle, than to let developers or testers discover them later and redo their work in order to fix them.
6.8.4 More Advice

Capers Jones offers the following advice to project managers:

"... all of the successful projects tend to follow a similar pattern even though they were created by different companies, in different countries, and within different subindustries, and had different business and technical purposes for creation... The pattern is this: there are myriad ways to fail when building large software systems. There are only a very few ways to succeed.

Jones indicates that all of the paths that lead to successful software include these twelve essential attributes:

Table 27: Capers Jones twelve essential attributes of successful software projects

<table>
<thead>
<tr>
<th>Attribute</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Effective project planning</td>
<td>7. Effective development processes</td>
</tr>
<tr>
<td>2. Effective project cost estimating</td>
<td>8. Effective communications</td>
</tr>
<tr>
<td>3. Effective project measurements</td>
<td>9. Capable project managers</td>
</tr>
<tr>
<td>4. Effective project milestone tracking</td>
<td>10. Capable technical personnel</td>
</tr>
<tr>
<td>5. Effective project quality control</td>
<td>11. Significant use of specialists</td>
</tr>
<tr>
<td>6. Effective project change management</td>
<td>12. Substantial volumes of reusable material</td>
</tr>
</tbody>
</table>

7. Case Studies

These case studies are a result of informal interviews of IT employees with a composite experience in four companies of various sizes and lines of business. The numerical data pertaining to each study is approximate, and its accuracy was not validated.

7.1 Case 1 – Requirements Analyst’s Account of Successful Project

This case study is an account of a software development effort related by the project’s Requirements Analyst (RA). The project was conducted May 1999 through April 2000 at a company that employs over 100,000 people worldwide. The company’s many lines of business include the provision of global IT services, management, consulting and business management to major corporate clients. The company employs 10,000-20,000 IT employees, and has probably 50-100 software development groups worldwide. This particular project involved generating metrics of system outage data and collecting the metrics in a database for reporting. Generating the metrics required the augmentation of an interface between an asset management system and a problem management system, and creating a process for transferring data to the reporting

---

database. The reports would then be used to benchmark future service level agreements between the company and its customer – a major telecommunications firm.

The project involved a matrixed team located mostly in Rochester, NY and Atlanta, GA, and had the luxury of a requirements analysis phase that was not time-limited. This was a project that had been undertaken and dropped several times over the previous five years. The requirements analyst had been told that the business requirements were too difficult to capture, and took on this project as a challenge.

The requirements definition process took five months. On a scale of 1-5, with 5 = best rating, the RA rates the overall requirements analysis effort a 4. She felt that business requirements were clearly defined. Although user requirements definition could have used some improvement, the RA felt that end users were sufficiently involved in the definition process, their tasks were sufficiently analyzed, and the product quality attributes were clearly defined. Although the requirements were not judged to be easy to understand, and were not represented pictorially, they were logically organized by functional area, mutually consistent and testable.

The requirements were reviewed and signed off by the customers, the resource manager, the project manager, the requirements analyst, and the software team’s technical lead. There was no representative from the quality testing group.

The project was not planned or scheduled until the completion of the requirements phase. Not including the 5 months for requirements gathering, the project took 4 months to plan, execute and test, and did not exceed the planned time line in spite of a 2-week delay during testing. The planned cost of the project was $600,000, which also was not exceeded.

During requirements elicitation, only one project resource was engaged – the requirements analyst. However, the requirements analyst had ample access to business and user representatives. Fifteen resources were engaged during project execution, however, another 10 business analysts and end users (not full-time) were added during the testing phase.

Although the system had undergone initial unit and some integration tests, written and run by developers, its initial pass rate of QA tests – including integration, regression and performance tests – was only 20%. The testing phase, which had been scheduled for 2 weeks, actually took 4. All errors were corrected prior to product release.
The RA believes that the quality of the resulting system was high, considering the fact that it had been considered an impossible dream for the previous 5 years. Customer feedback was also very positive, and the system continues to be used two years after release. The RA’s management was “very, very happy” with the product, and the project became “a bullet on the VP’s annual report to his management.”

The lesson here is that quality requirements elicitation leads to a successful project and a successful product.

### 7.2 Case 2 – Technical Lead’s Perception of Same Project

Case 2 recounts the project in Case 1 from the perspective of the project’s technical lead. The tech lead felt that although business requirements and functional specs were clear, user requirements were poorly defined and end users were not sufficiently involved in requirements elicitation. As a result, the product’s quality attributes were not clear. The tech lead has a high opinion of the requirements document in terms of organization, ease of understanding, testability, mutual consistence and lack of redundancy, but gives the overall requirements elicitation and analysis effort a rating of 3 out of 5.

The tech lead does not consider this project a success, and is not aware of the management’s opinion. She attests the lack of success to the fact that “users didn’t know what they wanted.” Her overall perception of the resulting product’s quality was 2 out of 5, (The RA’s rating here was 5.) and feels that the customers were “not happy” with the product.

Case 1 and Case 2 illustrate the subjective nature of assessing project and product success, and just how differently the same effort can be perceived based on one’s perspective.

### 7.3 Case 3 – Project Manger’s Account of Unsuccessful Project

This case involves a project described by its Project Manager (PM). The company is an internet computer-based training provider that employs about 600. Of the 600, about 150 employees are in software development, constituting about 5 development groups. The purpose of the project was to provide a custom reporting mechanism from a leveraged system to a customer account. Project dates were 11/13/2000 through 1/5/2001.

The requirements elicitation effort took 7 days – 2 days longer than planned. The PM felt that the business requirements were poorly defined (2 on a scale of 1-5) and user requirements were not understood (1 on a scale of 1-5). End users were not sufficiently involved in defining
requirements, but user tasks were sufficiently analyzed. Furthermore, acceptance criteria were not well defined or understood. However, the resulting requirements were logically organized, easy to understand, testable, mutually consistent and not redundant. The requirements document underwent a formal review and sign-off, which included the project manager, the requirements analyst, a software developer and the producer (business representative). The whole requirements analysis effort was rated a 3 on a scale of 1-5.

The project was completed on time. Five resources were initially allocated to the project; it took 7 to complete the project on time. The two additional resources were both QA testers. The test time was initially planned for 10 days. Management pressure resulted in shorting this time to 6 days, however, the figure then was changed to 10 again. The actual testing time was 10 days. The initial QA test pass rate was 90%, and all errors were fixed before product release.

The allocated and actual budget figures were not available to the PM, so it is not clear whether or not the project exceeded projected costs. Although the PM strongly feels that “users didn’t know what they wanted, even through they signed off on the business requirements”, the organization’s management considered this project to be a success because the project team “delivered what was agreed upon” within the allotted time. In addition, the management felt that the project met quality targets, even though they were poorly defined by the requirements analyst.

In the PM’s words, the customer was “not happy” with the product and is not using the functionality.

This is an example of a seemingly successful project resulting in an unsuccessful product. It is also an example of a project that was forced ahead in the interests of time, even though user requirements were not well understood.

7.4 Case 4 – Developer’s Account of Unsuccessful Project

This case was related by a software developer who doubled in the customer support role. The project was sponsored by a relatively small company, located in Rochester, NY and Belfast, ME, whose primary business was software development. The project dates were May 1995 through January 1996. At the time, the company employed a little over 50 people, the majority of whom (30+) were IT employees. The company handled one software project at a time.

The requirements elicitation phase for this project took 6 months, of which 4 was planned. The developer felt that the resulting business and user requirements were fairly clearly defined,
although the resulting functional specification could have been more clear. End users should have been more involved in requirements elicitation. User task analysis and product quality attributes were deficient. Requirements were average (3 out of 5) in terms of organization, ease of comprehension, testability, visual representation, mutual consistency and lack of redundancy.

The requirements document underwent a formal review and sign-off that did not include the customer, but did include the resource manager, project manager, requirements analyst, software developer, process analyst and a documentation specialist. Requirements were baselined and subject to change control, but basically followed a waterfall model.

Six months was initially allocated to the project, which took 9-10 months total. The cost overrun was roughly proportional -- $400,000 allocated, $600,000 actual. The $200,000 difference was billed to the client. The project did not meet quality targets, but was considered "relatively successful because it worked, but not 100%; also took longer than expected."

The developer feels that the testing effort was lacking. The testing was conducted both by developers and QA. The initial QA test pass rate was about 60%, and only about 85% of the errors were corrected. The testing phase was planned for 2 weeks, and took about 3 weeks.

The developer judged the quality of the product to be about a 3 on a scale of 1-5, while the customers considered it about a 2. The resulting system, sold to multiple corporate clients, ended up costing the firm between 2 and 5 large customer accounts over the following 2 years.

7.5 Case 5 – Same Developer’s Account of Successful Project

This case again involves a company that employs over 100,000 people worldwide, and whose many lines of business include the provision of global IT services, management, consulting and business management to major corporate clients. Again, this company employs 10,000-20,000 IT employees, and has probably 50-100 software development groups worldwide. The project purpose was to build an asset configuration management billing system for a major corporate client in the business of office imaging. The story was related by one of the project’s software developers, who also served as a requirements analyst and as subject matter expert in billing. The project in question took place between 1996 and 1998.

The first year of the project was planned for requirements elicitation, and coincides with the amount of time this phase actually took. The developer/RA felt that the business and user
requirements were very clearly understood and defined (5 out of 5), and the resulting functional specifications were also very clear (4 out of 5). He felt the end-users were sufficiently involved in eliciting requirements, the user tasks were well analyzed and the product quality attributes were clearly defined.

The resulting requirements were logically organized, easy to understand, testable, graphically represented, mutually consistent and not redundant. Acceptance criteria were clearly stated. Graphical representations included a context diagram, a process diagram, a “huge” entity relationship diagram, a user task diagram, a system map and a data flow diagram.

The requirements were formally reviewed and signed off by a team consisting of the resource manager, the project manager, the requirements analyst, software developers, a process analyst and a documentation specialist. After baseline, requirements were subject to change control in the context of a waterfall model. Overall, the developer/RA rates the requirements analysis phase a 5 on a 1-5 scale.

Project execution was planned for 1 year, but took about 1.5 years. The initial budget was estimated at about $15,000,000, but the actual budget came closer to $22,000,000. (Another participant in this project indicates that the budget figures were closer to $40,000,000, but confirms that the projected figure was exceeded.) The price includes labor, software licenses, agreement with out-of-the-box system vendor, and hardware costs. The planned number of resources was about 35, but the actual number was closer to 45. Because some resources were filling in from other teams, their time may have been paid for by Corporate, and may not have added to the cost of the project.

The developer/RA feels that quality targets were identified and met. The product underwent thorough testing by both developers and QA. The initial pass rate was about 70%, and all identified errors were corrected prior to product release. The testing phase was planned for 2 months, but took about 2.5 months.

The developer’s perception of the overall product quality is high (4 out of 5), and customer feedback was similar. They felt that as a result of this product, “billing accuracy greatly improved, and adjustment rate went way down.”

The lesson here is that a marginally successful project can still result in a well-received product, if the requirements are not rushed and are elicited with sufficient user participation.
7.6 Case 6 – Developer’s Account of Successful Iterative Development Project

This case was related by a software developer. Although the project dates are uncertain, the project took place about circa 1995, and involved about 4 months of requirements elicitation time, followed by 1-year’s iterative development effort by a development organization of about 150 people. The company is a Rochester-based payroll services company that at that time employed about 1,800 people, of whom about 200 were IT employees comprising about 5 software development teams.

The software developer feels that, although end-users could have participated more in the requirements elicitation and user tasks could have used more analysis, the business and user requirements were well understood, and the resulting functional specifications were clearly defined. The product’s quality attributes were clear.

The resulting set of requirements was logically organized by functional area, easy to understand, testable, visually represented, mutually consistent and not redundant. The graphical methods employed included a process diagram, an entity relationship diagram and a data flow diagram. Acceptance criteria were clearly stated.

The requirements were formally reviewed and signed off by a team that included the user, the customer, the resource manager, the project manager, the requirements analyst, software developers, testers, a process analyst and a documentation specialist. The initial set of requirements was baselined, but because the project followed an iterative development model, was subject to change throughout execution. On a scale of 1-5, the developer gives the requirements elicitation phase of this project a 4, overall.

Both the developer and his management considered this project a success. The project was completed on time. Although the developer is not know the allocated budget for the project, he “doubts” that the project was completed within the allotted budget constraints. Initial about 25 resources were allocated, however the actual number was closer to 40. Quality targets were identified and met. During testing about 1,000 defects were identified from 300 tests. About 90% of the defects were corrected. The testing phase overlapped with the development phase due to the iterative model, and took about 4 weeks, which is about the amount of time planned.
The resulting product was perceived as very successful by customers, and over the following 3 years, resulted in customer growth from 1,000 to about 10,000, as part of a planned roll-out. During those 3 years, fewer than 100 customers expressed dissatisfaction with the product.

The case illustrates the success of the iterative development model, and once again shows that a less than successful project can result in a highly successful product.

7.7 Case 7 – Requirements Analyst’s Account of Successful Project Resulting in Marginally Successful Product

This case involves a project to build an interface to dispatch trouble tickets between an IT infrastructure support system and a problem management system. The sponsoring company employs over 100,000 people worldwide. Its many lines of business include the provision of global IT services, management, consulting and business management to major corporate clients.

Again, this company employs 10,000-20,000 IT employees, and has probably 50-100 software development groups worldwide. Both the IT infrastructure support system and the problem management system wereleveraged for hundreds, if not thousands, of the company’s corporate customer accounts. The project team was matrixed, and involved development teams located in Rochester, NY, Plano, TX, Atlanta, GA and Raleigh, NC. The case is related by the requirements analyst, who also served in the roles of a project manager for the requirements phase, and participated in the testing and documentation efforts.

The time allotted to requirements analysis was not limited by a specific deadline. However, the goal was to include the functionality in the next planned release of the IT infrastructure support system. The requirements elicitation phase took about 2 months of non-continuous work effort on the part of the requirements analyst and the account’s business representative.

As with other case studies, the RA felt that end user participation was lacking and user tasks could have been better analyzed. Nevertheless, he also felt that business and user requirements were clearly defined and the resulting functional specifications was sound, although the product quality attributes could have been more clear.

The resulting requirements were logically organized, easy to understand, testable, mutually consistent and not redundant. Graphical representations of requirements were not employed until the design phase. The requirements underwent formal review and sign off by the customer account representative, the project manager, the requirements analyst and the software developer. Requirements were baselined and then subject to change control within the constraints of a
waterfall model. Once in development, the tight release schedule made changes difficult to propose and pass.

Including the 2 months of requirements elicitation time, the project took about 5-6 months. No expectations were set for the project time frame, so the project can be considered as having been delivered on time. The allocated or actual budget is not known, but the RA saw no evidence that it was exceeded. Although initial resource allocation figures are not known, additional developers were added based on technical assessment projections. This brought projected time down by 25% or more.

Although the project was considered successful by management, the RA is not convinced that it satisfied the needs of the users, primarily because they did not directly participate in the requirements elicitation efforts. The interface underwent extensive testing, and passed 80-90% of the tests on the first run. All identified defects were corrected prior to release. The testing phase went “a little longer than expected.”

The RA’s perception of the resulting product’s quality is average (3 out of 5). Customer feedback was mixed (also about 3 out of 5), primarily because there were different camps of customers, some of whom expressed positive opinions about the products, and some were negative.

This case illustrates the importance of user participation in requirements elicitation. Although the project was managed with a relatively high degree of competence, the success of the resulting product was marginal.
8. Conclusions and Implications

By means of a literature review and case studies, this paper has shown that the software development industry is prone to unsuccessful projects and to products of declining quality. The literature review and case studies demonstrate that the situation is largely attributable to poor understanding of user and business needs, requirements definition and project planning, which introduce unnecessary risk to software development projects. It is more the rule than the exception that newly engineered software products are immediately subject to enhancements. Frequently, an enhancement project precedes the initial release of the product. Such "enhancements" are usually essential functionality that had been overlooked in requirements analysis, or removed from project scope due to unforeseen cost and/or time overruns. The lack of success of software development projects brings into question the functional definition of a project itself.

The paper touches on a number of ways in which managers make unsuccessful software projects appear successful. These methods are confined within the constraints of the definition of a project – an undertaking that has a beginning and an end – and generally amount to redefining quality criteria after the fact, to make the product appear to have met the criteria. Perhaps a more useful and honest way of representing the issue would be to circumvent the confines of a "project", and create an environment of continuous development, where the software system is understood to be subject to changes. The success of either approach depends on a good understanding of business and user needs, how these needs translate into functional specifications, and how these needs and specifications change throughout the development effort.

The paper demonstrates the critical importance of requirements and usability analysis to the success of the project and the quality of the product. A number of sources indicate that the reasons for project impairment and failures are disproportionately attributed to insufficient user involvement, poor requirements analysis and management. Changing customer and project team expectations throughout the development effort are costly to the success of the project, and are particularly poignant at the completion of the project, when customers appraise the quality of the product. The seven case studies cited further demonstrate the difficulties faced in software projects.
Also repeatedly observed throughout this paper is the fact that software development and project management are not easy disciplines. The software project manager has a number of traditional project management methods and tools available to conduct a successful project. Unfortunately, the success of these methods is measured statistically, and following these methods does not guarantee successful results for the project at hand. Every “recipe for success” is actually a probability assessment based on previous “best practices” of experienced project managers. No two software projects are identical, so the best that adhering to past experience or a methodology can yield is a higher probably of success or a lower probability of failure. Many project managers don’t have an appreciation of statistics and methodology, and are more comfortable when they see programmers feverishly coding. Unfortunately, the main contribution of the “hurry up and code” mentality prevalent in today’s software development industry is increasing project risk, and virtually guaranteeing unpredictable results. As unintuitive as statistical evidence can be, the results of ignoring it are borne out in the poor success rate of today’s software development projects, and the poor quality of the resulting software.

The most important lessons a software project management can take away from this paper are that reduction of project risk is the most important factor in producing a predictable result, and that the key to reducing project risk is planning. This can only be accomplished with a thorough understanding of user and business needs, as well as sound requirements and usability analysis.
Annotated Bibliography


Technical report prepared for the SEI, presenting an approach to improving the usability of software systems by means of software architectural decisions. The report identifies specific connections between aspects of usability and software architecture.


Study of the speed and effect of growth of 80 Internet firms, that concludes that only 10% of these companies benefited from fast time to market. For the remaining 90%, moving too quickly is attributed to the company’s failure to develop a business plan, to respond to market conditions and ultimately to survive.


Tutorial on the Unified Modeling Language (UML), which is a graphical language for visualizing, specifying, constructing, and documenting software systems, starting with the system requirements.

Conallen, Jim, Building Web Applications with UML. Reading, Massachusetts: Addison-Wesley, 1999.

Guide to Web application development, written for project managers, architects, analysts, designers, and implementers. It focuses on UML – a common graphical language used for modeling software-intensive systems.


A collection of articles on best practices in requirement engineering focused primarily on software systems. The 35 articles cover current issues, basic terminology, and the phases of software requirements engineering, which include elicitation, analysis, specification, verification, and management.

Introduction to the UML and the application of its notation and semantics to the development of real-time and embedded systems.


Although the focus is on data warehousing, included is the role of requirements gathering in implementing data warehouse solutions, as well as an interesting aside on the distinction between data requirements and data modeling, and the importance of not using the two interchangeably.

Fowler, Martin, “The New Methodology.”
http://www.martinfowler.com/articles/newMethodology.html#N401, last viewed on 5/17/01

A synopsis of lightweight (or agile) methodologies for building software. The main objective of these methodologies is to minimize the amount of process involved in building software to a sufficiently useful level.


An introduction and description of Noriaki Kano’s method of analyzing requirements. The method helps evaluate product requirements’ impact on customer satisfaction, and system functionality and capability.


A warning against the use of paper prototyping as a means to gather user requirements. Instead, the following techniques are recommended: task analysis, surveys, focus groups, and contextual inquiry.


A project management primer that covers the four phases of a project’s life cycle; the three project parameters – quality, time, and cost; steps for controlling work in progress; and bringing the project to a successful conclusion through evaluation.


General description of various types of changes to be managed in software development projects, approaches and processes for managing changes, and an overview of software change management tools available on the market.

General description of the SCR (software cost reduction) requirements method, used primarily in high assurance systems, which include command and control systems, weapons systems, flight programs for commercial and military aircraft, control systems for nuclear plants, and most medical systems. Properties of high assurance systems include reliable service, security, safety, real-time capabilities, and fault-tolerance. Although the focus of the article is on a specific tool, the basic functions of this tool support four general principles of best practice software requirements analysis.

Herlea, Daniela Elena (Knowledge Science Institute, University of Calgary), “Users' Involvement in the Requirements Engineering Process”, http://spuds.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html

Focus on the requirements elicitation process, and the importance of involving end-users during this crucial phase of the software project. Although this article centers on a groupware system called TeamRooms, designed to facilitate communication between participants of a distributed requirements analysis team, the article also describes a number of basic methods of eliciting requirements.


Details of cost issues and ability to more effectively and realistically estimate the price of software development. Topics covered include metrics, estimation tools, object technology, global costs factors, development for multiple platforms, and the threat of litigation.


CD-ROM study guide designed to serve as preparation for the PMP Certification examination offered by the Project Management Institute (PMI). The focus of this resource is on general project management, which includes project scope management, time management, human resource management, cost management, risk management, quality management, procurement management, communication management and integration management.


Lexicon of ideas pertaining to software requirements analysis and problem solving, arranged in alphabetical order. The book’s central theme is the relationship of method both to problem structure and to description.

Description of how to gather and analyze software requirements using a process based on use cases, thereby reducing the incidence of duplicate or inconsistent requirements; communicating requirements that are understandable to both users and developers; communicating a vision of what the application needs to do without the distractions inherent in a coded prototype; and documenting the entire requirements process clearly and efficiently.


Suggestions on how to shorten e-project time to market by reducing and/or eliminating important quality planning steps from the project plan.


A primer on software project management as it pertains to the integration of enterprise software with existing systems. Focus on all phases of project management as well as post-integration considerations, which may be likened to evaluation of product success.


Concise statement of the reasons for and benefits of constructing software models prior to initiating building of the actual software system.


An argument against shortening product development time by tweaking individual phases of developing without considering the development cycle in its entirety. The author argues against shortening the development process. Evidence is cited that shortening concept development and design time results in delays later in the project and in overall cost and schedule overruns. In particular, the authors caution against moving the “quality gate” from the beginning of the development process to later phases.


An approach of capturing and prioritizing requirements continuously, which the author indicates shortens analysis phase of projects, controls scope creep, and increases customer involvement in system development.

Strategies, best practices and tips on how to shorten and control development schedules and keep projects moving. Also included is a discussion of exemplary practices and pitfalls, a list of classic mistakes to avoid and case studies to illustrate key ideas.


General purpose guide to managing software projects, aimed at those without formal project management training, in particular, top managers, executives, clients, investoris, end-user representatives, project managers, and technical leads. Areas of emphasis include planning, design, management, quality assurance, testing, and archiving.


A guide to methods of usability engineering and its leverage on product quality. Special emphasis on cost-effectiveness methods in developing user interfaces; avoidance of delay in software projects; pertinence of methods to particular stages of the development life cycle; and information on unique issues relating to informational usability.


Information on user-centered design, the technology life cycle, product design processes, and why it is so hard to design easy to use products which are successful in the marketplace.


Summary of a study by the CNSS to observe differences between traditionally managed, supplier-driven software development practices, and the emerging “preferred” way, which is market-driven and characterized by “early delivery of product to customers, innovation in technical approach, boldness in management approach, process of experimentation, decriminalization of defects, taking calculated risks, civility, push back, cooperation, collaboration, and competition.”

PMI guide to general project management, not specifically aimed at software development. Similar to the Kerzner guide to project management, the PMI guide covers project integration management, scope management, time management, cost management, quality management, human resource management, communication management, risk management and procurement management.


Description and case studies of software’s contribution to innovation, as well as innovative approaches to software development. Innovative approaches to software development management covered include: individual inventor-innovators; small interactive teams; and monitor programs.

Rational e-Development Company, Improving Software Economics, presented in Rochester, NY by Walker Royce on 6/12/01.

A seminar outlining four ways in which iterative software development improves software economics – reducing complexity, improving the process, improving team proficiency and automating with tooling – with the help of Rational software development tools.


A description of the “U Effect” as a phenomenon of changing customer satisfaction over the life of a program, from “high” at the beginning, to “low” at the midpoint during detailed design, back to “high” at delivery. The “U Effect” is caused by unmet expectations and the lack of a working relationship between the customer and the supplier at the beginning of a project or program. The article proposes a number of ways in which project or program managers can deal with the “U Effect”.


Definitive text on requirements analysis, covering ‘best practice’ methods, models, and techniques for the requirements analysis practitioner.


A user guide and reference manual describing formalized processes that comprise requirements gathering.

Description of approaches to mitigate project success in time-sensitive projects. Basic advice is to define “quality” as it pertains to less-than-ideal circumstances, where perfection is out of the question.


A white paper stressing the importance of customer input into product development, and offering techniques of engaging customers throughout the product definition and development process.


A somewhat dated but broad range of requirements analysis techniques, including: the software life cycle; translating a real-world problem into a technical solution; various modeling techniques, including software system modeling, process modeling, data/process separation, data independence, data/process models, data modeling approaches, and object-modeling approaches; analyzing a requirements specification; determining system scope; designing a solution; documentation; data dictionary; entity relationship modeling; functional dependency and normalization; designing a client/server environment for Oracle an SQL server; event-activity analysis; dataflow diagrams for process analysis; techniques for process design; object-oriented design considerations; and CASE tools.


A record of a 1995 study conducted by the Standish Group to assess the scope of software project failures, the major factors that cause software projects to fail, and the key ingredients that can reduce project failures.


A follow-up to the 1995 CHAOS Chronicles study. In the author’s words, “From November 6th through the 9th, 1995, The Standish Group held CHAOS University in Chatham, Massachusetts. CHAOS University was a follow-up to the CHAOS study published in January 1995. The Standish Group estimates that almost 80,000 projects were cancelled in 1995. While The Standish Group identified the ten main causes of these failures (along with possible solutions), it was unclear whether these solutions could be implemented. In response to this and by invitation only, CHAOS University brought together 60 IT professionals for the purpose of digging down to create two additional levels of detail in the implementation of the success factors.”

An article describing why it is important for IT projects to meet delivery commitments, and the top six reasons why IT projects tend to fail to meet commitments.


User Interface Engineering is a Massachusetts-based consulting company specializing in web usability engineering. The site contains a number of articles and white papers on many topics relevant to software usability and other web design issues.


An interesting case study describing the use of a paper prototyping technique.


Techniques for managing the requirements engineering process throughout the development cycle, including tools to facilitate communication between users, developers and management. It provides advice on how to stop requirement scope creep, manage change requests, set project realistic project expectations, cut revisions and costs, and to improve software products.


General article about requirements analysis which provides a working definition of “requirements”; categorizes requirements analysis; offers a suggested requirements development process; describes “best practices” in requirements analysis; advises on how to enact “best practices”; and touches on ways to “sell” requirements analysis to IT managers.


Exploration of a design methodology that involves users early in the process to describe business events, that are then partitioned into logical, more easily managed segments, resulting in a conceptual model that reflects real business needs and accelerates the delivery process. It also includes a description of functional point estimation of time and cost of future software projects based on system requirements.

Web site about Extreme Programming approach to software development. It contains useful information about the ideas behind Extreme Programming, appropriate conditions for using this approach to software projects and “lessons learned”.

Xprogramming.com  http://www.xprogramming.com/

A web site self-described as “a center for information about Extreme Programming, the exciting new streamlined software development process originally described by Kent Beck. XProgramming.com is offered as a community resource for those interested in XP and related topics.”


Managerial and technical issues that determine the success or failure of a project, providing a framework and process to help keep projects on track and ensure that requirements are addressed properly throughout a project’s life cycle.


Frequently-referenced book that describes a prevalent approach to software development that defies traditional project management principle of crash points -- theoretically insurmountable constraints that cannot be overcome by extra hours of work in the absence of other resources, such as additional time, budget or human resources. This book provides practical advice on dealing with the politics of such work situations, including negotiation techniques, dealing with human resource issues, applicable processes, tools and technology.
### 9. Appendix A – Requirements Elicitation Methods

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Application Design (JAD)</td>
<td>An example of representative design, is a workshop approach to system design which has become the most common user involvement methodology in North America. The JAD approach is based on communication through documentation, fixed requirements and rules of work enforced through methods. Today, meetings are conducted using CASE tools: for depicting data-flow diagrams, Entity-Relationship diagrams, state transitions and other diagramming techniques, screen painters, … [and] groupware.</td>
</tr>
<tr>
<td>Participatory Design (PD)</td>
<td>An example of representative design, is the &quot;Scandinavian approach&quot; to system development. PD focuses on much stronger involvement of the users than JAD does, facilitating a mutual learning process between users and designers, and joint experiences into a simulated work situation.</td>
</tr>
<tr>
<td>Inquiry-Based Requirements Analysis Model</td>
<td>&quot;A series of questions and answers designed to pinpoint where information needs come from and when&quot;90</td>
</tr>
<tr>
<td>Inquiry Cycle Model</td>
<td>A &quot;formal structure for describing discussion about requirements&quot;, addresses the case of mass-market-driven product development, for which there may be no clear customer authority.</td>
</tr>
<tr>
<td>AMORE (Advanced Multimedia Organizer for Requirements Elicitation)</td>
<td>A prototype modeling environment which includes methods and techniques used in the area of requirements elicitation. AMORE uses the System for Access to Information and Learning (SAIL) to provide assistance to elicitors, designers, customers, anyone who need to examine and manipulate requirements.</td>
</tr>
<tr>
<td>Soft System Methodology (SSM)</td>
<td>A methodology for &quot;soft&quot; problems, the ones which deal with 'what' of the problem situation, not with 'how', like &quot;hard&quot; problems. Checkland's methodology recognizes groups who decide on requirements and negotiate contractual issues between customers and suppliers. It suggests managing this kind of process by guided intervention in meetings when people with different objectives and perceptions discuss and provide the environment for understanding of the problem at hand.</td>
</tr>
<tr>
<td>ORDIT</td>
<td>ORDIT focuses on the need of the organization as opposed to the individual, and agrees on the importance of social context in the requirements definition process. The process has four broad interactive component subprocesses: scoping, modeling, requirements capture and solution option</td>
</tr>
<tr>
<td>Nature</td>
<td>A large ESPRIT project initiated in Aachen, Germany. NATURE defines a framework which builds three specific theories. The requirements domain theory gives advice on what context knowledge is relevant and how to organize it. The requirements process theory offers a unified process meta model in which a small set of building blocks covers a larger spectrum of process guidance strategies with more flexibility than other software process or workflow models. The knowledge representation theory aims at defining what domain and process knowledge to capture, and how to manage this knowledge using an effective mix of informal, semiformal and formal representations. 91</td>
</tr>
</tbody>
</table>

---

90 In Hearlea, [http://spuds.cpsc.ucalgary.ca/KAW96/herlea/FINAL.html](http://spuds.cpsc.ucalgary.ca/KAW96/herlea/FINAL.html), attributed to a study by Potts, Takahashi and Anton 1994. Not able to regain access the site as of 5/25/01

91 In Hearlea, [http://spuds.cpsc.ucalgary.ca/KAW96/herlea/FINAL.html](http://spuds.cpsc.ucalgary.ca/KAW96/herlea/FINAL.html), attributed to a study by Jarke, Pohl, Domges, Jacobs and Nissen, 1995. Not able to regain access the site as of 5/29/01
10. Appendix B – Data, Process and Object Modeling Techniques

The Robertsons’ textbook approach to requirements analysis\(^{92}\) outlines the following components of thoroughly-defined and analyzed requirements:

1. Analysis models
2. Data flow diagrams
3. A variety of viewpoints
4. Data viewpoint
5. Data models
6. Data flow diagrams
7. Leveled data flow diagrams
8. Current physical viewpoint
9. Data dictionary
10. Essential viewpoint
11. Event-response models
12. Mini specifications
13. Modeling new requirements
14. New physical viewpoint\(^b\)
15. Object-oriented viewpoint

A software requirements textbook published by the The Learning Tree International\(^{93}\), breaks requirements analysis into data analysis, process analysis and object analysis. Each approach is outlined in terms, of the following steps:

**Analyzing Data Requirements (entity-relationship modeling):**
1. Crow’s foot notation
2. Entity subtypes and supertypes
3. Optional and mandatory participation
4. Binary, n-ary, and recursive relationships
5. Determining entity states
6. Chen and other related notations

**Process Analysis**
1) Event-Activity Analysis
   a) Event-activity modeling
   b) Determining events at the system boundary
   c) Stimulus/response
   d) Use Cases
   e) Cross-reference matrix
   f) Quality checks

2) Dataflow Diagrams for Process Analysis
   a) Dataflow diagrams
   b) Leveled dataflow diagrams
   c) Expanding dataflow diagrams
   d) Top-down design

---


e) Events analysis  
f) Ensuring consistency between levels  
g) Quality checks  
h) Other quality issues  

3) Techniques for Process Design  
a) Process design  
b) Structure chart  
c) Transform centered design  
d) Pseudocode  
e) Stepwise refinement  
f) Procedural stepwise refinement  
g) Action diagrams  
h) Action diagram nested hierarchically  
i) State transition diagrams  
j) Using state transition diagrams to model the states of a control process  
k) Decision tables  

Object-Modeling Techniques  
1) The object-oriented paradigm  
2) Abstract and concrete classes  
3) Object identity  
4) Links and objects  
5) Associations  
6) Many-to-many associations  
7) Whole-part structures (aggregation)  
8) Whole-part structures  
9) When should you use aggregation?  
10) Collections  
11) Operations  
12) Primary operations  
13) Non-primary operations  
14) Deduction  
15) Identifying operations using object life histories  
16) States of objects  
17) State diagrams  
18) Message connections  
19) Generalization  
20) Inheritance  
21) Polymorphism  
22) Information hiding  
23) Encapsulation  
24) Message (event) traces  
25) Use cases
11. Appendix C – Case Study Interviews – Questions and Answers

<table>
<thead>
<tr>
<th>General Org Questions</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many people are in your immediate software development organization?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 1-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 10–30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 30–50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>- 50+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>150</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>What is your role within the organization? (select all that apply)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Resource manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Project manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>- Requirements analyst</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Software developer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Software tester</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Process analyst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>- Documentation specialist</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>- Other (please specify)</td>
<td></td>
<td></td>
<td>Tech Lead</td>
<td>cust suppt</td>
<td>billing SME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Where is your immediate organization located?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Matrixed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- City</td>
<td></td>
<td></td>
<td></td>
<td>Roch, Belfast ME</td>
<td></td>
<td>Roch</td>
<td>all teams in Roch</td>
</tr>
<tr>
<td>How large is your company?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Number of employees:</td>
<td>100,000</td>
<td>600</td>
<td>50+</td>
<td>100,000</td>
<td>1,800</td>
<td>100,000</td>
<td></td>
</tr>
<tr>
<td>- Number of IT employees:</td>
<td>20-30,000</td>
<td>150</td>
<td>30+</td>
<td>25,000</td>
<td>200</td>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>- Number of software development groups:</td>
<td>30</td>
<td>5</td>
<td>1</td>
<td>30</td>
<td>5</td>
<td>100+</td>
<td></td>
</tr>
<tr>
<td>Is software development your organization’s primary business?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>- Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Definition Questions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Think of the last software project you completed. How much project time was initially allocated to requirements definition and analysis for this project?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Duration:</strong> No limit  No limit  5 days  4 mos  1 yr  4 mos - iterative development  not limited - object was to get functionality into next SW release</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements Definition Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>How much project time did requirements definition and analysis actually take?</td>
</tr>
<tr>
<td><strong>Duration:</strong> 5 mos  5 mos  7 days  6 mos  1 yr  4 mos  2 mos, not continuous</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>On a scale of 1-5, rate the level of your agreement with the following statements about requirements definition:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business requirements were clearly defined</strong>  4  4  2  4  5  4  4</td>
</tr>
<tr>
<td><strong>User requirements were clearly defined</strong>  3  1  1  4  5  4  4</td>
</tr>
<tr>
<td><strong>Functional system requirements were clearly defined</strong>  3  4  3  3  4  4  4</td>
</tr>
<tr>
<td><strong>End-users were sufficiently involved in defining requirements</strong>  5  don't know  1  3  4  3  2</td>
</tr>
<tr>
<td><strong>User tasks were sufficiently analyzed in defining requirements</strong>  4  don't know  4  2  4  3  3</td>
</tr>
<tr>
<td>Case 1</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td><strong>Product quality attributes (acceptance criteria) were clearly defined</strong></td>
</tr>
<tr>
<td>On a scale of 1-5, rate the level of your agreement with the following statements about requirements analysis for this project:</td>
</tr>
<tr>
<td><strong>Requirements were logically organized into functional areas</strong></td>
</tr>
<tr>
<td>- Requirements were easy to understand</td>
</tr>
<tr>
<td>- Requirements were testable</td>
</tr>
<tr>
<td>- Requirements were visually represented by graphical models</td>
</tr>
<tr>
<td>- Requirements were not redundant</td>
</tr>
<tr>
<td>- Requirements were mutually consistent (i.e., no conflicting requirements)</td>
</tr>
<tr>
<td>- Acceptance criteria were clearly stated</td>
</tr>
</tbody>
</table>

If requirements were modeled graphically, please indicate which of the following techniques were used:

- Context diagram
- Process diagram
- Entity relationship diagram
- User task diagram
- System map
- Data flow diagram
- Other: (please specify)

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>y</th>
<th>n</th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, in design</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n/a</td>
</tr>
<tr>
<td>?</td>
<td>y</td>
<td>n</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>y</td>
<td>y</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

Did the requirements document undergo a formal review by project stakeholders?

- Yes
- No

If so, which of the following parties was involved?

- x
- x
- y
- y
- x
- x
<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource manager</td>
<td>x</td>
<td></td>
<td>y</td>
<td>y</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Project manager</td>
<td>x</td>
<td></td>
<td>y</td>
<td>y</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Requirements analyst</td>
<td>x</td>
<td></td>
<td>x</td>
<td>y</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Software developer</td>
<td>x</td>
<td></td>
<td>y</td>
<td>y</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Software tester</td>
<td>x</td>
<td></td>
<td>x</td>
<td>n</td>
<td>n</td>
<td>x</td>
</tr>
<tr>
<td>Process analyst</td>
<td></td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Documentation specialist</td>
<td></td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Other (please specify)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Customer rep/Producer</td>
<td>n</td>
</tr>
</tbody>
</table>

Did the requirements document undergo a formal sign-off prior to start of design/implementation?

- Yes | x | x | x | y | y | x | x |
- No

If so, which of the following parties was involved?

- User | x | x | x | x | x | x |
- Customer | x | x | n | n | n | x |
- Resource manager | x | x | y | y | x | x |
- Project manager | x | x | y | y | x | x |
- Requirements analyst | x | x | x | y | y | x | x |
- Software developer | x | x | y | y | x | x |
- Software tester | x | n | n | n | n | x |
- Process analyst | n | y | x | y | y | x |
- Documentation specialist | x | y | y | n | x | x |
- Other (please specify) | Customer rep/Producer | n | n | n | n |

Were requirements baselined upon sign-off? | Y | Y | Y | x | x | x |
Were requirements subject to change control? | y | y | y | x | x | x |
On the scale of 1-5, how would you rate the quality of the requirements analysis phase for | 4 | 3 | 3 | 3 | 5 | 4 | 3 |
<table>
<thead>
<tr>
<th>Project Management Questions</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>What were the start and end dates of the last completed project participated in?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dates do not reflect add'l 1 yr of req dekn</td>
<td>Includes 4 mos req analysis</td>
<td>Includes 2 mos requirements analysis</td>
</tr>
<tr>
<td>• Start Date:</td>
<td>May-99</td>
<td>May-99</td>
<td>11/13/2000</td>
<td>May-95</td>
<td>Jun-97</td>
<td>1 yr</td>
<td>5-6 mos</td>
</tr>
<tr>
<td>• End Date:</td>
<td>Apr-00</td>
<td>Apr-00</td>
<td>1/5/2001</td>
<td>Jan-96</td>
<td>Jun-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In your opinion, was this project a success?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>x</td>
</tr>
<tr>
<td>• Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Comments:</td>
<td>&quot;functionality is used to this day&quot;</td>
<td>&quot;users didn't know what they wanted&quot;</td>
<td>&quot;users didn't know what they wanted, even though they signed off on business reqs&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Did your management consider this project a success?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>x</td>
</tr>
<tr>
<td>• Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Comments:</td>
<td>&quot;Don't know&quot;</td>
<td></td>
<td>&quot;relatively successful because it worked, but not 100%; also took longer than exp&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How much time was initially allocated to this project?</td>
<td>4 mos, not including req analysis</td>
<td>&quot;Don't know&quot;</td>
<td>&quot;Delivered what was agreed upon&quot;</td>
<td>6 mos</td>
<td>1 yr + 1 yr reqs</td>
<td>1 yr</td>
<td>5-6 mos</td>
</tr>
<tr>
<td>• Planned Duration:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Planned Start Date:</td>
<td>May-99</td>
<td></td>
<td>11/13/2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Planned End Date:</td>
<td>Sep-99</td>
<td></td>
<td>1/5/2001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was the project delivered within expected time frame?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>x</td>
</tr>
<tr>
<td>• Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 1</td>
<td>Case 2</td>
<td>Case 3</td>
<td>Case 4</td>
<td>Case 5</td>
<td>Case 6</td>
<td>Case 7</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Actual Duration:</td>
<td>9 mos</td>
<td>9-10 mos</td>
<td>1.5 yrs + 1 yr reqs</td>
<td>1 yr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Start Date:</td>
<td>May-99</td>
<td>11/13/2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual End Date:</td>
<td>Apr-00</td>
<td>1/5/2001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td></td>
<td>&quot;Don't know&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>What budget was initially allocated to this project?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocated Dollar Amount:</td>
<td>600,000</td>
<td>&quot;Don't know&quot;</td>
<td>&quot;Don't know&quot;</td>
<td>400,000</td>
<td>15,000,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td>&quot;There were time delays in reqs &amp; testing, but the expensive phases were not altered, so project was within cost constraints.&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was the project delivered within expected budget constraints?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;doubt it&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Dollar Amount:</td>
<td>600,000</td>
<td>22,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percent of Planned Dollar Amount:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td>&quot;There were time delays in reqs &amp; testing, but the expensive phases were not altered, so project was within cost constraints.&quot;</td>
<td>&quot;Don't know&quot;</td>
<td>&quot;Don't know&quot;</td>
<td>&quot;$200,000 overrun was billed to the customer&quot;</td>
<td>&quot;includes labor, software licenses, Peregrine agreement, equipment&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How many resources were initially allocated to this project?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of resources</td>
<td>1 during req analysis; 15 more during execution</td>
<td>&quot;Don't know&quot;</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was the project delivered within expected resource constraints?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 1</td>
<td>Case 2</td>
<td>Case 3</td>
<td>Case 4</td>
<td>Case 5</td>
<td>Case 6</td>
<td>Case 7</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual number of resources</td>
<td>25</td>
<td>x</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td>add'1 10 BA's and end users pulled in for testing</td>
<td>&quot;Don't know&quot;</td>
<td>add'2 testers were added</td>
<td>&quot;some resources were from other teams, so may have been paid for by Corporate, not from project budget&quot;</td>
<td>developers were added to reduce projected time based on tech assessment -- reduced by 25% or more</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Were quality targets for this project identified?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td></td>
<td>&quot;Don't know&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Did the project meet quality targets?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td>&quot;not successful; one req was not useful&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;Project was deemed successful, but not sure it satisfied business needs&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**Quality Testing Questions**

- **Was the product tested?**
  - Case 1: y
  - Case 2: y
  - Case 3: y
  - Case 4: y
  - Case 5: y
  - Case 6: y
  - Case 7: y

- **Who wrote system tests?**
  - Developers
  - Developers & QA
  - QA
  - QA & customer

- **Who ran system tests?**
  - Developers
  - Developers & QA
  - QA
  - QA & customer

- **What was the pass rate?**
  - 20%
  - 90%
  - 60%
  - 70%
  - 1000 defects from 300 tests

- **On scale of 1-5, what is your assessment of the system test effort?**
  - 2-Jan
  - 3
  - 4
  - 4
  - 4
  - 4

- **Were system tests based on requirements?**
  - y
  - y
  - y
  - y
  - y
  - y
  - y, but more from design

- **Were they exclusively based on requirements?**
  - n
  - n
  - n
  - n
  - n
  - n

- **Was there unit testing?**
  - y
  - y
  - y
  - y
  - y
  - y
  - y, some, not enough

- **Who performed it?**
  - Developers
  - Developers
  - Developers
  - QA & Developers
  - Developers
  - Developers
  - Developers

- **Was there regressions testing?**
  - y
  - y
  - y
  - y
  - y
<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who performed it?</td>
<td>QA</td>
<td>QA</td>
<td>QA</td>
<td>developers</td>
<td>QA</td>
<td>QA, req analyst &amp; customer rep</td>
</tr>
<tr>
<td>Was there integration testing?</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Who performed it?</td>
<td>QA</td>
<td>QA</td>
<td>QA</td>
<td>developers &amp; QA</td>
<td>QA &amp; developers</td>
<td>QA, req analyst &amp; customer rep</td>
</tr>
<tr>
<td>Was there performance testing?</td>
<td>y</td>
<td>y</td>
<td>?</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>Who performed it?</td>
<td>QA</td>
<td>QA</td>
<td>QA</td>
<td>developers &amp; QA</td>
<td>developers &amp; QA</td>
<td>developers</td>
</tr>
<tr>
<td>Were all test cases consistent with acceptance criteria?</td>
<td>y</td>
<td>&quot;Don't know&quot;</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>What percentage of test cases passed on first run through?</td>
<td>20%</td>
<td>&quot;Don't know&quot;</td>
<td>&quot;Don't know&quot;</td>
<td>60%</td>
<td>70%</td>
<td>don't know</td>
</tr>
<tr>
<td>Were all mistakes corrected prior to product release?</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>no - 85%</td>
<td>y</td>
<td>n - 90%</td>
</tr>
<tr>
<td>What was the planned duration for testing?</td>
<td>2 wks</td>
<td>&quot;Don't know&quot;</td>
<td>10, then 6, then 10 days</td>
<td>- 2 wks</td>
<td>2 mos</td>
<td>4 wks, includes overlap due to iterative devpt</td>
</tr>
<tr>
<td>What was the actual duration?</td>
<td>4 wks</td>
<td>&quot;Don't know&quot;</td>
<td>10 days</td>
<td>- 3 wks</td>
<td>2.5 mos</td>
<td>4 wks</td>
</tr>
<tr>
<td>Product Quality questions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a little longer than exp'd</td>
</tr>
<tr>
<td>On a scale of 1-5, what is your overall perception of the resulting product's quality?</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3-Feb</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Did your organization receive any feedback from the customers of the product?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Yes</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>- No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments:</td>
<td>&quot;very very happy&quot; &quot;project was a bullet on VP's annual report to his mgt&quot;</td>
<td>&quot;not happy&quot;</td>
<td>&quot;not happy&quot;</td>
<td>&quot;billing accuracy greatly improved; adjustment rate went way down&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>If so, rate the customer perception of the product's quality on a scale of 1-5.</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2-4 -- different camps of customers had different opinions</td>
</tr>
<tr>
<td>Question</td>
<td>Case 1</td>
<td>Case 2</td>
<td>Case 3</td>
<td>Case 4</td>
<td>Case 5</td>
<td>Case 6</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Of your new customers, how many chose your product based on referrals from existing customers?</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td># customers greater than 1,000 to 10,000 over 3 yrs, as part of planned roll-out</td>
</tr>
<tr>
<td>If your product is subscription-based, how many subscription renewals has it generated?</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>How many existing customers did your organization/company loose as a result of the release of the new product?</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>2-5 large customers over following 2 yrs</td>
<td>none</td>
<td>less than 100 over 3 yrs of roll-out</td>
</tr>
</tbody>
</table>