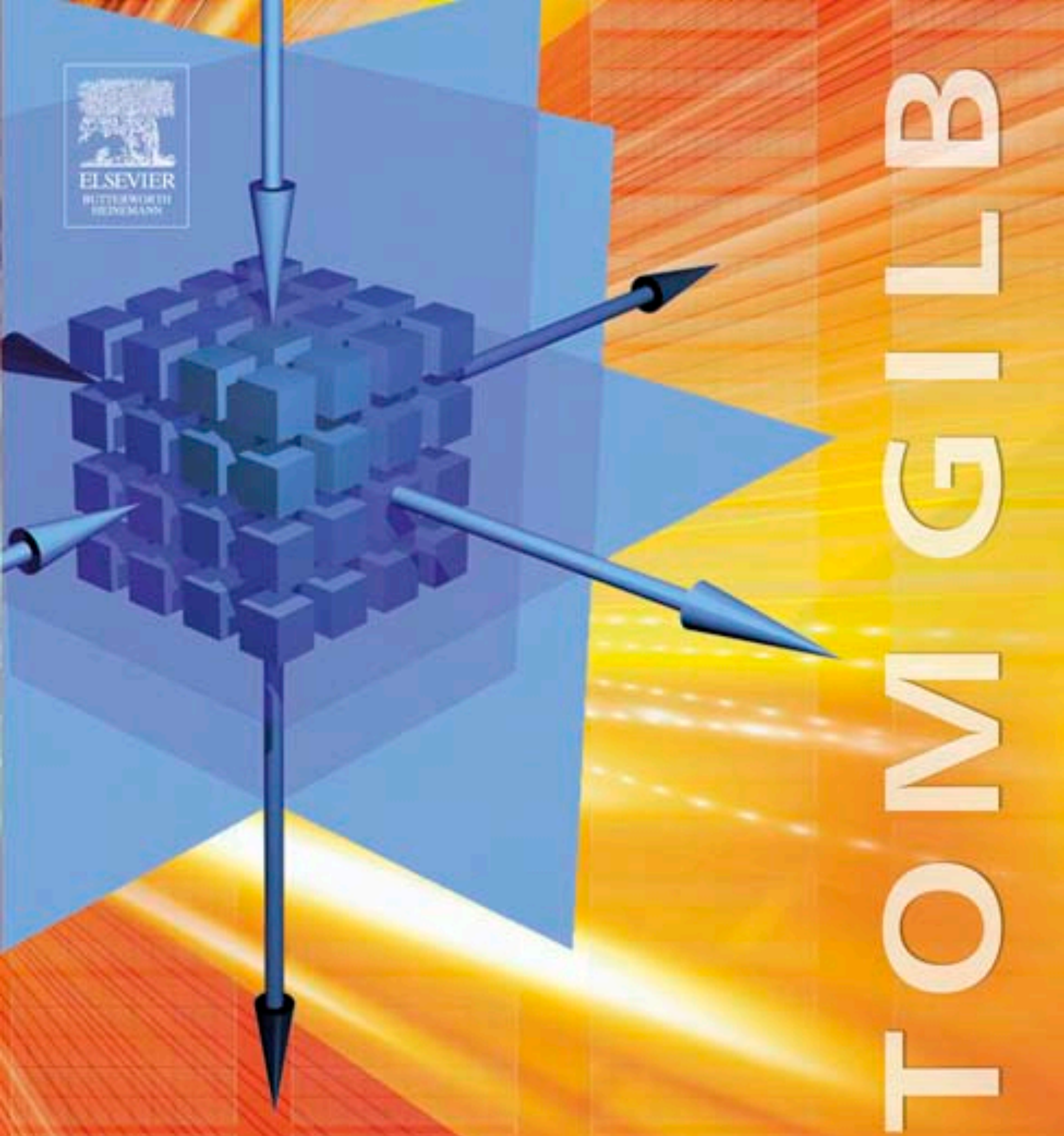


COMPETITIVE ENGINEERING

A HANDBOOK FOR SYSTEMS ENGINEERING, REQUIREMENTS ENGINEERING, AND SOFTWARE ENGINEERING USING PLANGUAGE

COMPETITIVE ENGINEERING

A Handbook for Systems Engineering Requirements Engineering, and Software Engineering Using Planguage

COMPETITIVE ENGINEERING

A Handbook for Systems Engineering Requirements Engineering, and Software Engineering Using Planguage

Tom Gilb
Email: Tom@Gilb.com
URL: www.Gilb.com

Editor: Lindsey Brodie,
Middlesex University, UK
Email: lindseybrodie@btopenworld.com

ELSEVIER
BUTTERWORTH
HEINEMANN

AMSTERDAM • BOSTON • HEIDELBERG • LONDON • NEW YORK • OXFORD
PARIS • SAN DIEGO • SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Elsevier Butterworth-Heinemann
Linacre House, Jordan Hill, Oxford OX2 8DP
30 Corporate Drive, Burlington, MA 01803

First published 2005

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication Data
A catalogue record for this book is available from the Library of Congress

ISBN 0 7506 6507 6

Typeset by Integra Software Services Pvt. Ltd, Pondicherry, India
www.integra-india.com
Printed and bound in Great Britain

CONTENTS

To Jane

Foreword

Competition is the strongest force shaping today's product development landscape. The race to create customer and end user value is intense. In addition, companies large and small face ever-increasing product complexity, pressure to reduce time to market and increase productivity, and unprecedented challenges from globalization.

Competitive Engineering contains powerful tools to apply to these problems. At the same time, the tools are both practical and simple – a rare combination. Over the last decade, I personally have applied these tools in a variety of settings in software development and more general product development, on projects of various sizes. Thousands of students have been through training and workshops I have authored that contain Planguage, Evo, Specification Quality Control and other facets of Competitive Engineering. The vast majority of students immediately recognize their value and go on to use these beneficially on projects. *Competitive Engineering* is based on decades of practical experience, feedback and improvement, and it shows. This stuff works.

To be effective over a wide range of problems, a method or tool must possess many qualities: flexibility, scalability, portability and learnability, to name a few. The methods and tools found in *Competitive Engineering* are up to the challenge. They are designed to be tailored to local culture and practices. The central ideas are so fundamental that they apply to a huge range of project types and sizes. And, while they are rich enough that they require serious study to master, they can be learned and used effectively by almost anyone; I have taught them to people in product development, service delivery, manufacturing, site construction, information technology, eBusiness, quality, marketing and management.

You may encounter some resistance when first proposing or teaching these concepts; I have, and still do from time to time. Don't be discouraged. These are often revolutionary concepts in relation to a group's existing practices. Keep in mind that it is not necessary to use everything you find in *Competitive Engineering* right from the start. Instead, use Competitive Engineering to create Competitive Engineering. Start using Evo to improve things in small steps based on what is most valuable in your environment. Use Planguage to document stakeholder needs, success criteria and the like. Before you know it, you will have made significant progress. It all feels remarkably natural once you get going.

Erik Simmons
Intel Corporation
Requirements Engineering Practice Lead
Corporate Quality Network

ENDORSEMENTS

When I was a young engineer, I thought all system problems were merely a matter of applying clever technological solutions. "If we had better technology," I thought, "we'd be able to get this problem solved quickly."

After more than 30 years in the IT and systems world, I now know better. Creating, building, and delivering high quality computer-based systems, on-time and within budget not only requires solid technology, but also a meaningful process, effective project management, comprehensive risk control, and broad-based communication between all constituencies at all levels of the project. In fact, it is the last item in this list that is probably most important (and the most difficult to achieve). Projects fail because communication fails.

System and software engineers and their customers need a consistent mechanism for communicating the purpose of the system, the constraints that must be addressed, the design and implementation strategies that are to be applied, the risks to be managed, and the measures of quality that are relevant and meaningful. But how do we achieve a consistent mechanism for communication and how do we use this to better manage and implement complex systems?

In this book, Tom Gilb provides us with answers that are both elegant and comprehensive. He describes the Planguage method – "a practical set of ideas, to help you get better control over all forms of planning, design, engineering, and project management."

To be honest, those of us who are industry veterans have heard this before. Literally dozens of methods (and books) have made similar claims and then failed to deliver. Why then, should we believe that Planguage is any different? There are many reasons.

Gilb has designed an evolutionary approach that will allow you to define system requirements clearly and unambiguously. More importantly, he provides a bridge that enables you to describe the resultant solution, design a high quality implementation, and then analyze how the proposed solution impacts business objectives. In addition, he stresses quantitative evaluation, so that progress toward the competitive goals of your system can be evaluated competitively. Planguage enhances communication – at the specification level, at the design level, at the project level, and at the process level. Projects succeed when communication succeeds, and Planguage leads to successful communication.

As Tom himself admits, this book is not a light read. Competitive Engineering provides thought-provoking ideas on almost every page, is rich in detail, and comprehensive in scope. It provides guidance for every system engineering activity and a thorough description of every aspect of Planguage. You'll have to spend time with Gilb's ideas, but once you understand and apply them, your ability to engineer complex systems will be greatly enhanced.

Roger S. Pressman, Ph.D.
President, R.S. Pressman & Associates, Inc.
e-mail: pressman@rspa.com

Competitive Engineering stakes out unusual ground in engineering literature. The ground it takes is doubly unusual for a systems engineering book. *Competitive Engineering* works to provide pragmatic and directly applicable methods to fundamental design problems that cross application domains. Where many books provide procedural methods that apply only to particular problems, *Competitive Engineering* seeks to be general and to be applicable to many forms of complex systems. Where many books take on abstract or theoretical aspects of system development, *Competitive Engineering* provides specific, directly applicable techniques. And where many books address only narrow business models, such as contracted development, *Competitive Engineering* applies to a range of evolutionary and competitive developments.

Tom Gilb clearly intends for the book to be taken as a whole. Planguage provides a documented and uniform set of concepts and terms that a team can use to organize development efforts from simple to complex. As Tom explains, the book can be viewed as a comprehensive handbook to managing the development of complex systems. Its background is particularly strong for information systems, but it also applies to many other types of systems.

However, systems engineers should rightly be interested in this book even if they have no intention of adopting Planguage in its entirety. *Competitive Engineering* contains many other important nuggets, in consequence of the unique ground it has staked out, that are important to systems engineering even removed from full adoption of Planguage. A few examples are the sections on Scales of Measure, Impact Estimation, and Evolutionary Project Management.

One of the intellectual foundations of systems engineering is decision theory. Decision theory rests on our ability to discover a proper set of attributes on which to measure the goodness or worth of a system, to quantify relative to those attributes, to evaluate the consequences of design choices relative to those attributes and to make decisions based on those attributes. *Competitive Engineering* adds a rich set of heuristics and methods to the notoriously difficult problem of discovering good sets of attributes, quantifying performance on complex attributes and communicating the consequences of design choices relative to those attributes. The approach in *Competitive Engineering* is strongly pragmatic, while also being theoretically grounded. A user who has committed to Planguage can take the approach as a whole and find strong guidance for fully implementing it. A user who has not adopted Planguage as a whole can also benefit, because he or she will discover that the Scales of Measure and Impact Estimation concepts are solidly grounded and can be lifted and transferred into other development ontologies and other decision theory methods. The 'lift-and-carry' into other methods will benefit from the strong heuristics and well-thought-out communication methods presented in the book.

In a similar way, *Competitive Engineering*'s take on Evolutionary Project Management is highly useful on its own. Virtually all software-intensive systems today are developed in an evolutionary way, even when we don't plan to. While the importance and basic mechanisms of evolutionary development are well known, the community is in need of wider sets of guidelines and alternatives for actual implementation. It is in this area that *Competitive Engineering* delivers important new material.

Systems engineers should find *Competitive Engineering* widely useful, with or without the additional framework provided by Planguage. Even without adopting Planguage as a whole there are numerous important principles and techniques that can benefit any system project. And those who dip in looking for solutions to one problem or another may come to appreciate the full framework of Planguage.

Dr Mark W. Maier
Distinguished Engineer at The Aerospace Corporation and Chair of the INCOSE Systems Architecture Working Group. Co-author of *The Art of Systems Architecting*, Second Edition (CRC Press).

Preface

Background to writing *Competitive Engineering*

It has been sixteen years since *Principles of Software Engineering Management* (Tom Gilb, 1988, Addison-Wesley) was published. Since then I have continued to develop Planguage and many changes have been introduced. So, my main intention in writing *Competitive Engineering* (CE) is to document the current basic definition of Planguage (that is, the language and its methods).

In practice, the discipline of writing this book has also caused considerable improvement in the consistency of ideas and the quality of the explanation. Hopefully, readers will forgive me that the style of this book is deliberately 'less chatty' than *Principles of Software Engineering Management*. The aim is to provide a fundamental systems engineering handbook, which is more directly concerned with providing practical guidance on how to use Planguage.

In large part, CE is based around the Glossary of Planguage. The Glossary gives additional rigor to the book, as it has been applied to the entire text, including the Glossary itself.

Major influences on Planguage

The dominating influences behind the creation of Planguage include:

- The works of Deming, Juran, Crosby, Jevons (*The Principles of Science*, Dover Edition, 1960, originally published 1875), Boehm, Weinberg, Lord Kelvin, Keeney, Koen and Peters. See also the Bibliography and the citations in this book.
- My work with *real* engineers and managers in the industrial world; who want and need these ideas. It is amazing to me to see how each new piece of work, or consulting, directly results in evolutionary improvements to the theory and practice of Planguage.

- My many professional friends, clients and students, who appreciate, encourage, comment and discuss Planguage and share with me their ideas, case studies, papers and books.

> See http://zapatopi.net/Kelvin/quotes.html, which reads: ''In physical science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it. I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be.''
>
> *Lord Kelvin*

How to use this book

I do not expect readers to adopt everything in this book. Adoption must evolve based on feedback from real use.

Only *generalized* processes and rules are given in this book, and so it is quite likely that they will need to be *tailored* to *your* organization. In any case, some of the rules and processes in this book are rather too long for everyday use. This is because explanatory text has been included. If you intend to use any of these rule sets, then some editing is required to produce a shorter 'working' version for Specification Quality Control (SQC) purposes. (You can always make reference from your shortened version to other text for more background information.)

You are at liberty to adopt and adapt any of the CE ideas to your needs.

Some book conventions

Terminology: I decided to use the following as the main terms[1] in Planguage:

- 'Requirement specification' and 'requirement engineering' rather than 'requirements specification' and 'requirements engineering', respectively.

[1] The alternative terms are declared as synonyms.

- 'Function' as an adjective, rather than 'functional' (so Planguage uses 'function requirement' and 'function specification').
- I retained 'systems engineering' as it is such a widely used term (even though I do feel it needs to be 'systems' engineering' with an apostrophe!).
- 'Resource' has been used as a main collective attribute term, rather than 'cost'. This was a difficult decision because 'resource' in the USA tends to only mean 'staff'. Planguage usage is wider: 'resource' includes all committed money, staff, time and any other assets.

Formatting of dates: To avoid the text 'ageing', most of the dates have been declared as user-defined terms, such as 'Next Year.' In practice, users should use more precise dates, such as 'December 15, 2004', or 'Initial Delivery plus 3 Months'.

In many examples within the book, the use of the 'Version' parameter, and other administrative parameters, have simply been omitted to reduce complexity of the examples.

Planguage Glossary: Approximately 180 concepts have been formally defined, exemplified and annotated in the CE Glossary. The other 75 per cent of defined Planguage concepts are and will be in the complete glossary on the web. They are selected from over 640 defined Planguage concepts. The complete and updated Planguage concept glossary will be found at www.gilb.com. At the beginning of each chapter are listed the key glossary concepts.

The reader is well advised to consult the Glossary when trying to make sense of the text. In fact, it might be a good idea for the reader to scan through the entire Glossary, stopping at interesting concepts and getting a sense of the types of concepts I have defined.

I often think that the main lasting contribution of this book lies in the Glossary itself. It is not that I imagine the entire world standardizing on these terms! Rather, that I hope this Glossary might be quite useful in helping to develop improved standard systems engineering concepts. I do not doubt that we can all make these concept definitions even better. However, the reader can be assured that all the concept definitions were arrived at after considerable struggle and due consideration of many points of view and needs! Not least was the struggle to make them internally consistent.

At the very least, I hope that the Glossary helps the reader to be sure of what the text is saying. I also hope that a study of the Glossary will give the student an excellent grounding in systems engineering concepts.

Tom Gilb, Kolbotn, Norway
Email: Tom@Gilb.com

ACKNOWLEDGEMENTS

I would especially like to thank my clients too numerous to mention, both the companies and the enthusiastic individuals, and classroom students. They created the need, and validated the practice, for so many of the Planguage ideas.

Specific credit must also go to the following:

My personal editor, Lindsey Brodie: she helped edit *Principles of Software Engineering Management* (Tom Gilb, 1988, Addison-Wesley) and has worked with me as a consultant. She has edited all my papers for many years. She has done a heroic and wonderfully conscientious task of editing this book, undoubtedly changing unintelligible, well-intended prose into consistent, intelligible text and diagrams. The majority of the diagrams are hers alone in concept and execution. If the book is readable, it is Lindsey who deserves the credit. If it is not, then I must have insisted on something she disagreed with!

The following have been our core team of advisors for this book. We could always trust them to have interesting, useful opinions and to do some useful research for us. They saved us from many a fatal error! They all practice, and teach, what we preach here too!

Don Mills (New Zealand), especially for his discussion on implicit direction of numeric values for levels on scales of measure.

Erik Simmons (Portland, Oregon), Intel Corporation for permission to use two extracts from his teaching material on 'Landing Zone', and who always has a sensible, practical answer to everything.

Stuart Woodward (Ipswich, England) who decided to teach himself Planguage, method by method, and provided the most interesting feedback and insights on what happened when he successfully applied the methods.

My son, Kai Thomas Gilb has been a consistent influence in keeping the Planguage ideas simple and straightforward. He has an absolutely independent mind: we constantly teach and consult together, and constantly argue the technical points. He does not give up until I see the light. He has written his own book manuscript (in a more popular style than CE) and developed two automated tools for Planguage.

Thanks too must go to numerous others for pointing out various issues and problems in Planguage and in the text. Too many to recall all of them.

Of course, any remaining errors in this book are entirely the author's responsibility!

Planguage continues to evolve, and it never fails to amaze me how new Planguage ideas continue to emerge. However, Planguage has reached a stage where publication is timely. I hope that readers will enjoy the book, and find something within it that they can apply – tomorrow, if not today!

Further acknowledgements

- Clarkson N. Potter for the Alice and The Cheshire Cat and the Humpty Dumpty figures.
- Gerrit Muller, Embedded Systems Institute, Eindhoven for permission to use his Stakeholder diagram.

INTRODUCTION

Competitive Engineering (CE) is about technological management, risk control, and breakthrough improvement in complex business systems, projects and processes. It is systems engineering, with application to all forms of planning, requirement specification, design and project management. It also applies to management of organizations, both top management and technical management. 'Competitive Engineering' is hopefully the end result of using this book's ideas.

What is in *Competitive Engineering*?

CE is taught using 'Planguage.'[1] Planguage consists of a new industrial systems engineering language for communicating systems engineering and management specifications, and a set of methods providing advice on best practices. 'Planguage' is central to CE and permeates all themes of this book.

The **Planguage Specification Language** is used to describe all the requirements, designs and plans for a system.

The main **Planguage Methods** are as follows:

- **Requirement Specification**: used to capture all the different requirement types. Emphasis is placed on specifying competitive performance and resource attributes quantitatively.
- **Impact Estimation**: used to evaluate designs against the requirements. It is also used during project implementation to track progress towards meeting the requirements.
- **Specification Quality Control**: used at any stage of a project to check the adherence of any plan, contract, bid or technical specification to best practice specification standards.
- **Evolutionary Project Management**: used to plan and monitor implementation of the selected designs.

The reader will, hopefully, find that these are all very practical and innovative methods, compared with current practice and literature.

[1] The word 'Planguage' is derived from a combination of 'plan' and 'language.' It is pronounced like 'language' with the initial 'p' pronounced as in 'plan.'

How was *Competitive Engineering* developed?

The Planguage language and methods have gradually emerged from my practical experience, since 1960, as a teacher and consultant to industry. Since the early 1980s, earlier versions of the central Planguage elements have been adopted both by pilot projects and/or corporate-wide in several of my client multinationals including HP, IBM, Intel, Philips, Nokia, Ericsson and Douglas Aircraft (now Boeing). The interest and practical acceptance has encouraged me to expand and refine the definition of the language well beyond my earlier books and papers.

In the past, my public courses, books and articles have explained Planguage, under a variety of names and in terms of its various subsets, exclusively for software engineering purposes. However, since the early 1980s, Planguage has been used in top management and various systems engineering disciplines. This book has been written to reflect that fact. The aim is that this book is useful for any 'systems engineering' or 'systems engineering management' purpose.

What is special about Planguage?

Planguage integrates all the basic systems engineering disciplines from requirement specification to product delivery. It has at its core the aim of delivering the stakeholder-critical values. It makes the technical strategies clearly subordinate to the required results. It gives us a detailed set of ways to express our ideas for a system, including very many useful glossary-defined concepts, such as: objective, strategy, design and risk.

Planguage provides a common language for all the different disciplines to communicate with each other. It enables interdisciplinary groups to work as real teams towards well-documented common purposes.

Planguage is designed for adaptation and tailoring to your own specific projects, organizations and cultures.

Planguage lays the groundwork for systematic learning organizations and continuous work process improvement. It is based on the fundamental principles of feedback, of 'Plan-Do-Study-Act' cycles of activity as taught by W. Edwards Deming (Deming 1993) and Joseph Juran (Juran 1974). These are probably the most powerful weapons we have for improving productivity and economics. It is also oriented towards the ideas of Philip B. Crosby on defect prevention and 'clear measurable requirements' (Crosby 1996).

Planguage gives us tools to tackle large and complex systems in a more systematic way than current common practice. Used properly, it should reduce the high risk of waste, delay and failure, which has plagued all large-scale modern projects and 'high-tech' disciplines for decades. We have plenty of good, intelligent and well-intentioned people; now they, hopefully, have a useful tool for better understanding and management of the fast-moving, complex and 'not-yet-experienced' world, which seems to be our current daily working environment.

CE's practical ideas are already proven in practice internationally in electronics engineering, telecommunications engineering, aircraft engineering, top management, marketing, information technology and industrial software engineering projects. Once they are imbedded in corporate practice, they stay there. Once the individual knows and uses these ideas, they are irreversibly sold on the practicality and usefulness of these methods.

Planguage should be viewed as a powerful way to develop and implement strategies that will help your projects to deliver the required competitive results.

How to use *Competitive Engineering*

This is not a book to be read quickly and forgotten. Some CE chapters could expand to book length to fully explain their concepts. Indeed, this is the first in a series of related books, some of which are already written in draft form.[2]

This book is more like a dictionary or a handbook. It is intended to be a unifying standard. It should serve, for years, as the basis of your professional development (as it has for me and for my clients). Study it as needed. Try out the ideas in practice. Study more detailed literature. Translate it into your organization's local dialect. Use it to make rapid progress towards putting in place additional teaching and improved standards for your engineering and management methods.

[2] See www.Gilb.com for initial draft samples.

Structure of *Competitive Engineering*

The key contents of *Competitive Engineering* are as follows:

- **Chapter 1: Planguage Basics and Process Control**: This chapter explains why Planguage is necessary and introduces the structure of the Planguage language and methods.
- **Chapter 2: Introduction to Requirement**: This chapter outlines the fundamentals of requirement specification: the framework for setting targets and constraints.
- **Chapter 3: Functions**: This chapter describes functions and function requirements.
- **Chapter 4: Performance**: This chapter describes the basics of how to specify performance attributes quantitatively.
- **Chapter 5: Scales of Measure**: This chapter discusses finding and defining appropriate scales of measure.
- **Chapter 6: Resources, Budgets and Costs**: This chapter outlines how to specify the resource requirements.
- **Chapter 7: Design Ideas and Design Engineering**: This chapter describes how to find and specify design ideas. It also outlines the Design Engineering process.
- **Chapter 8: Specification Quality Control**: This chapter gives an overview of the Specification Quality Control method (also known as 'Inspection'), which measures specification quality against your own tailored specification standards.
- **Chapter 9: Impact Estimation**: This chapter describes the Impact Estimation method, which is used to evaluate quantitatively the impact of design ideas on your performance and resource requirements, and can also be used to track quantitatively project progress towards critical objectives.
- **Chapter 10: Evolutionary Project Management**: This chapter gives an overview of Evolutionary Project Management. It gives you the basic concepts of evolutionary delivery and discusses how you identify evolutionary steps.
- **Glossary**: The glossary in the book has over 180 concepts. It provides detailed reference definitions, supporting the text in the chapters (only the main concepts could be fitted into this book, the additional concepts can be found in the complete glossary at http://www.Gilb.com/).

Format of *Competitive Engineering*

The format of each chapter is the same:

- **Section 1: Introduction**: Each chapter has an introduction that puts the chapter in context.

- **Section 2: Practical Example**: Each chapter has a simple example that aims to introduce the subject area of the chapter.
- **Section 3: Language Core**: Each chapter has the basic new Planguage concepts described.
- **Section 4: Rules**: Each chapter (apart from Chapter 8, Specification Quality Control) includes some rules, which can be used as a specification standard for the chapter's subject.
- **Section 5: Process Description**: Each chapter has a process definition that corresponds to the subject of the chapter.
- **Section 6: Principle**: Each chapter has 10 principles, which are intended (in a light-hearted manner) to highlight and remind you of the key ideas discussed within the chapter.
- **Section 7: Additional Ideas**: Each chapter discusses some advanced ideas to try to give you some insights beyond the basic ideas described in the chapter.
- **Section 8: Further Example/Case Study**: Each chapter contains a more detailed example. Real case studies or practical examples from the author's first-hand experience are used.
- **Section 9: Diagrams/Icons**: Each chapter has diagrams providing graphical ideas for presenting the chapter content. Specifically, the icon language supporting Planguage is shown.
- **Section 10: Summary**: Each chapter is summarized in the last section.

A friendly warning

This book is intentionally written in a very condensed style. Don't get discouraged if you have to slow down to understand it, or if you have to reread parts. It is 'useful ideas per hour' which count, not 'pages turned per hour'.

Chapter
1

Planguage Basics and Process Control

The Purpose of Planguage

Glossary Concepts

Planguage
Standards
Rule
Process
Procedure
Task

1.1 Introduction: Why We Need a Different 'Systems Engineering' Approach

As the rate of technological change has 'heated up,' I am sure we have all seen that, increasingly, nobody 'knows all the answers.' Previously we could rely on comparatively *stable* environments (technology, workforce, experienced people, politics and economics). People knew how to solve problems because they had solved similar ones before. In addition, the concept of learning by apprenticeship was valid; 'masters' could pass on their wisdom over a time span of years.

Things are currently moving so fast that it is dangerous to assume there is any first-hand knowledge of the technology we are going to use, or of the markets we are going to sell to. Even the organizational and social structures that we are targeting are constantly changing. Authors such as Tom Peters have long since clearly documented these trends and threats (Peters 1992).

So we have to find out 'what works now' by means of practice, not theory. We need to develop things in a different way. We have to learn and to change, faster than the competition.

The fundamental concepts needed now in systems engineering include:

Learning through Rapid Feedback

Feedback is the single most powerful concept for successful projects. Methods that use feedback are successful. Those that do not, seem to fail. Feedback helps you get better control of your project, by providing facts about how things are working in practice. Of course, the presumption is that the feedback is early enough to do some good. This is the main need: rapid feedback.

Dynamic Adaptability

Projects have to be able to respond to feedback and also to be able to keep pace weekly or monthly with changing business or organizational requirements. Projects must continuously monitor the relevance of their current work. Then they must modify their requirements and strategies accordingly. Any product or organizational system should be continuously evolving or it dies. Coping with external change *during* projects and adapting to it *during* projects is now the norm, not the exception. Stability would be nice, but we can't have it!

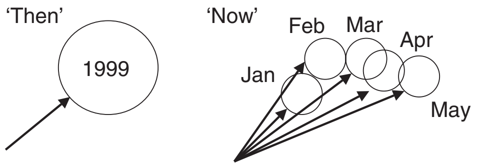


Figure 1.1
Our requirements are changing faster due to external changes.

Capturing the Requirements

It is true of any system that there are several *Critical Success Factors*. They include both performance requirements (such as serviceability, reliability, portability and usability) and limited resource requirements (such as people, time and money). Projects often fail to specify these critical requirements adequately:

- not *all* the critical success factors are identified
- no target *numeric* values for survival and success are stated
- variations in targeted requirements for differing times and differing places, are not addressed:
    - the effect of peak loads, or system growth, on the required levels, is not taken into account
    - the concept of very different attribute levels, being required by different parts of the system, or by different stakeholders, is not considered
- no practical ways to measure the results delivered to stakeholders are specified alongside the requirement specification.

The result is that our ability to manage successful value delivery is destroyed from the outset. It is impossible to engineer designs to meet non-specified or ambiguous requirements. It also is impossible to track changes for such ill-specified requirements.

Focus on Results

The primary systems engineering task is to design and deliver the best technical and organizational solutions, in order to satisfy the stakeholders' requirements, at a competitive cost. Projects must ensure that their focus is on delivering critical and profitable results. Albert Einstein is quoted as saying: "Perfection of means and confusion of ends seem to characterize our age."[1] Unfortunately, this still appears true today. It is the delivery of the required *results* from a system that

[1] Calaprice, Alice [Editor]. 2000. *The Expanded Quotable Einstein.* Princeton University Press. ISBN 0-691-07021-0.

counts. The process used and the technology selected are mere tools in the service of the results.

Interdisciplinary Communication

Clear communication amongst the different stakeholder groups is essential. Common problems include:

- ambiguity, due to specification that lacks precise detail
- critical specifications being 'lost' in overwhelming detail
- technical specification being unintelligible to the management, who reviews it
- inadequate tracking of specification credibility: its source, status and authorization level.

Leadership and Motivation

Clear vision makes a huge difference. Clear vision gives a common focus for logical decision-making. When people understand the overall direction, they tend to make good *local* decisions. Only the critical few decisions need to be made at the top. It is important for all team members to be able immediately to channel their energies in a true common team direction.

Receptiveness to Organizational Change

It is also important for system engineers to know that their organizational culture really supports improvement in systems engineering methods. In other words, that people are actively encouraged to look for improvements and to try out new solutions. Positive motivation can be everything! It is not a case of demanding improvement, more a case of supporting and rewarding people who seek it.

Continuous Process Improvement

The quality guru, W. Edwards Deming considered that: "Eternal process improvement, the Plan-Do-Study-Act (PDSA) cycle, is necessary as long as you are in competition." Having best-practice systems engineering standards in place, measuring conformance to them and continually trying to improve them is necessary if you are to compete well.

The only thing that *should* not change is a great change process.

Practical Strategies for Systems Engineering

Planguage[2] (the specification language and methods described in this book) aims to support all the above concepts with practical ideas and methods; it has numerous practical strategies for projects to adopt.

In-built in all these Planguage strategies is risk management. Handling of risk is fundamental to Planguage. I do not believe that risk management should be a separate discipline. We can deal with risks better when we do so in every detailed specification and plan, and in every systems

Practical Strategies for Risk Management

1. **Quantify requirements**: All critical performance and resource requirements must be identified and quantified numerically.
2. **Maximize profit, not minimize risk**: Focus on achieving the maximum benefits within budget and timescales rather than on attempting to eliminate all risk.
3. **Design out unacceptable risk**: Unacceptable risk needs to be 'designed out' of the system consciously at all stages, at all levels in all areas, for example, architecture, purchasing, contracting, development, maintenance and human factors. This means selecting lower-risk options.
4. **Design in redundancy**: When planning and implementing projects, conscious backup redundancy for outmaneuvering risks is a necessary cost.
5. **Monitor reality**: Early, frequent and measurable feedback from reality must be planned into your development and maintenance processes, to identify and assess risks before they become dangerous.
6. **Reduce risk exposure**: The total level of risk exposure at any one time should be consciously reduced to between 2% and 5% of total budget.
7. **Communicate about risk**: There must be no unexpected risks. If people have followed guidelines, and are open about what work they have done, then others will have the opportunity to fight risks constructively. Where there are risks, then share that information.
8. **Reuse what you learn about risk**: Standards, and other forms of work process guidance, must capture and assist good practice. They must be subject to continuous process improvement.
9. **Delegate personal responsibility for risk**: People must be given personal responsibility in their sector for identification and mitigation of risks.
10. **Contract out risk**: Make vendors contractually responsible for risks. They will give you better advice and services as a result.

Figure 1.2
Practical strategies for risk management.

[2] Pronounced like 'language' with a 'p' as in 'plan.'

implementation process. Figure 1.2 gives a list of strategies for risk management. All these strategies can be found in some aspect of Planguage.

1.2 Practical Example: Twelve Tough Questions

Here are some probing questions for controlling risk. They are powerful tools, which will help you in your everyday work. I call them the 'Twelve Tough Questions' – see the next page.

These 'Twelve Tough Questions' will help you find out 'what people *really* know.' They will help you find out how strong a foundation their opinions and recommendations are based on. From the answers to these questions – or maybe the *lack* of answers – you can see risks; and what needs to be done to reduce them. Try asking these questions when you next review a proposal, or at your next decision-making meeting. You will probably see the power of them immediately. Get your *management* to ask these questions.

Copy this next page (permission granted as long as you include copyright). Carry it with you to your next meeting or frame it on your wall. Use it to arrest fuzzy thinking in your company and client documents.

Twelve Tough Questions

1. **Numbers**
   Why isn't the improvement *quantified*?
2. **Risk**
   What is the degree of *risk* or uncertainty and *why*?
3. **Doubt**
   Are you *sure*? If not, *why* not?
4. **Source**
   Where did you get *that* information? How can *I* check it out?
5. **Impact**
   How does *your* idea affect *my goals and budgets*, measurably?
6. **All critical factors**
   Did we forget anything critical to survival?
7. **Evidence**
   How do you *know* it works that way? Did it '*ever*'?
8. **Enough**
   Have we got a *complete* solution? Are *all* requirements satisfied?
9. **Profitability first**
   Are we planning to do the '*profitable* things' first?
10. **Commitment**
    Who is responsible for failure, or success?
11. **Proof**
    How can we be *sure* the plan is working, *during* the project, *early*?
12. **No cure, no pay**
    Is it '*no cure, no pay*' in a contract? Why not?

© Tom Gilb 2000–5
A full paper on this is available at www.Gilb.com

1.3 Language Core: Planguage Basics and Process Control

Planguage consists of a specification language and a corresponding set of process descriptions (or methods). The Planguage *language* terms are used together with the Planguage *processes* for specification, analysis, design (also called planning, engineering, architecture or problem-solving) and management of processes, projects or organizations.

Planguage Specification Language

The specification language (usually called simply 'Planguage') is used to specify requirements, designs and project plans. Planguage consists of the following elements:

- **A set of defined concepts**. The Planguage Glossary contains the master definition of concepts as used within Planguage (Examples of concepts: objective, goal and function).
- **A set of defined parameters and grammar**. The Glossary also contains the set of defined Planguage parameters (Examples of parameters: Scale and Meter) used for specification.

  The grammar consists of Planguage syntax rules. These syntax rules are given in this book by example, rather than being formally stated. The aim is to show 'best known practice' of how the Planguage parameters should be specified to be useful. Note the examples given are only 'reasonable examples,' the reader should feel free to add to them, to improve them and to tailor them.

- **A set of icons**. Each icon is used for graphical representation of a specific Planguage concept and/or parameter. Icons may either be keyed in on a keyboard, or drawn. *For example, <fuzzy angle brackets> are used to indicate a 'poor' definition in need of improvement and, the icon, '< >', is in the Glossary under 'Fuzzy'.*

Relevant subsets of the Planguage language are introduced throughout the book in the Language Core section of the chapters. More formal definitions can also be found in the Glossary.

Planguage Process Descriptions

The Planguage process descriptions (or methods) provide recommended best practice for carrying out certain tasks. The reader should consider these defined processes as useful 'starter kits', but should plan to extend, improve and tailor them to their needs, purposes and experiences. The set of Planguage process descriptions is as follows:

- **Requirement Specification** (**RS**). (*See Chapter 2 and sub-processes in Chapters 3, 4, 5 and 6*)
- **Design Engineering** (**DE**). Design Engineering is concerned with identifying, selecting and sequencing delivery of design ideas (*see Chapter 7*)
- **Specification Quality Control** (**SQC**). SQC is used for evaluating the quality of any technical document and, for identifying and preventing defects (*see Chapter 8*)
- **Impact Estimation** (**IE**). IE is used for evaluating designs and monitoring the impact of results on the goals and budgets. It plays a central role in Design Engineering (*see Chapter 9*)
- **Evolutionary Project Management** (**EVO, also known as Evo**). Evo is used to deliver results in a series of high-value (highest value/best benefit to cost ratio delivered earliest), small (say, less than 2% of total project development time) evolutionary steps (*see Chapter 10*).

Note:

1. *SQC measures the degree to which a specification follows its specification rules. It directly measures the 'loyalty to engineering standards.'*
2. *Impact Estimation and Evolutionary Project Management measure the power of the design ideas in the marketplace.*

The process descriptions for the above methods can be found in the Process Description section of the relevant chapters.

Standards

As Tom Peters pointed out in *Liberation Management* (Peters 1992), the only remaining reason for having a very large organization is to share 'know-how' about best practices. Standards are an important form of sharing such know-how. (Other examples would be patents, market knowledge and specific customer knowledge.)

Standards can be termed 'Work Process Standards.' They can be usefully classified into specific types of guidance as follows:

- Policies
- Rules
- Process descriptions
- Forms and document formats
- Defined concepts (such as found in the Planguage Glossary)
- Language conventions (such as Planguage grammar)
- Work rates (such as 'checking rate')
- Others.

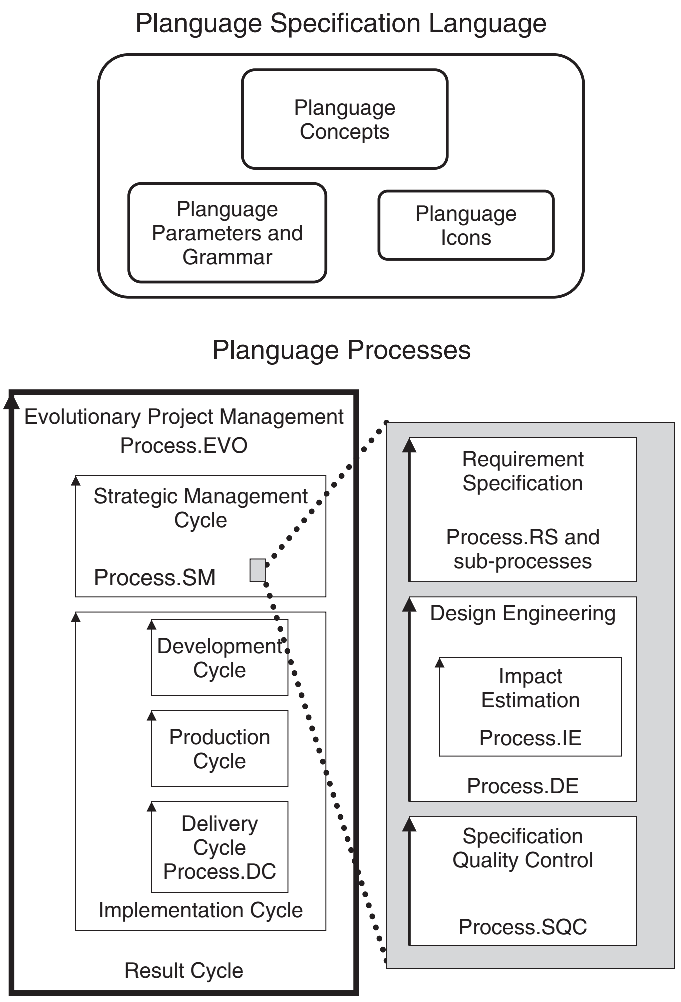

Planguage Specification Language
Planguage Concepts
Planguage Parameters and Grammar
Planguage Icons
Planguage Processes
Evolutionary Project Management Process.EVO
Strategic Management Cycle
Process.SM
Development Cycle
Production Cycle
Delivery Cycle Process.DC
Implementation Cycle
Result Cycle
Requirement Specification
Process.RS and sub-processes
Design Engineering
Impact Estimation
Process.IE
Process.DE
Specification Quality Control
Process.SQC

Figure 1.3
Diagram of the components of Planguage. Even more detailed, and more correct (as a consequence of being able to use the feedback from practical experience and, from any information being up to date) specification of the requirements and design ideas is likely to occur *within* the frequent development cycles. Detailed explanation of the Evo result cycles can be found in Chapter 10 (and in the glossary).

For specific examples, see Sections 1.4, 'Rules' and 1.5, 'Process Description.'

Standards can be *generic*, or can be tailored to *specific* tasks (for example, to contracting, testing, writing and installation) and tailored to *specific* stakeholder environments (for example, sub-supplier, novice, top management and customer).

Standards must be *brief* and *non-bureaucratic*. I favor, as a basic rule, limiting standards to *one-page* in length. I have found that my clients stick to a one-page format, finding it very practical. When there is only one page, detail cannot overwhelm people. For example, only the 20 to 25 or so 'most important' standards ideas can fit on a page (to be adopted, new standards must force bad ones 'off the page').

Standards must change as experience dictates. The *owners* of the standards must update the standards specification when better practices are discovered, so that new knowledge is shared and is rapidly put into use. People should be taught and motivated to use the standards, unless they can justify otherwise.

Rules

Rules are standards that provide specific guidance to follow when carrying out a process. They are also used in Specification Quality Control (SQC) to define and detect major defects in a specification. Individual rules should justify their presence in standards by the *potential resource savings* that can be expected from using them.

Process Descriptions

Process descriptions (or methods) are standards that describe the best practice for carrying out work tasks. The process format used for Planguage process descriptions consists of three basic elements:

- **Entry Conditions** – to determine whether it is wise to start the procedure
- **Procedure** – specifying for a task what work needs to be done and how best to do it
- **Exit Conditions** – to help determine if the work is 'truly finished'.

Entry Conditions

It is not good enough to allow employees to simply plunge into a work process and 'just do it.' The conditions must be right for success, not ripe for failure. Entry conditions are a list of what an organization has learned are the necessary prerequisites for avoiding wasted time and, for avoiding the failure of a specific work process.

Before beginning any procedure, its entry conditions must be checked. If the entry conditions are not met, then starting the procedure is high risk. It is likely to be better to remove the negative conditions before proceeding. Entry conditions should be built on experience of what is high risk and high cost.

Procedure

A procedure is a sequenced list of instructions, describing how to carry out the task. It documents the current recommended best practice.

EXAMPLE P3: For each design idea, estimate its numeric impact on the Scale of all the attributes.
P4: Continue identifying/specifying or refining design ideas until the specified safety margin is reached.

Exit Conditions

Exit Conditions are used to evaluate if the task is reliably and economically completed. They specify the safe and economic conditions for exit from a process to a 'next' process. Exit conditions are also built on experience from previous releases to the next work process.

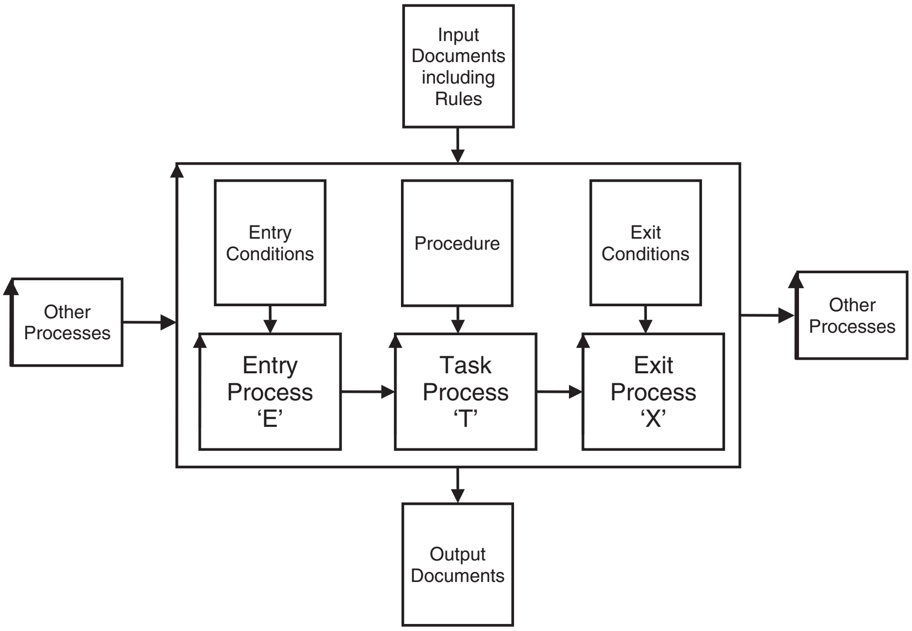


Figure 1.4
Diagram of a simple process showing its sub-processes and its relationship to other processes and documents. The input documents for each process include the rules, the entry conditions, the procedure and the exit conditions. The diagram also shows how the 'ETX' concept for a process is derived. A rectangle is the symbol for a 'written document.' A rectangle with arrow is a 'process' symbol. An example of such a process could be 'Requirement Specification.'

Table 1.1 Description of some of the main generic Planguage parameters, concepts and icons.

Basic Planguage Parameters, Concepts and Icons

Concept or Parameter	*Meaning*	*Used for*	*Note also*
Planguage Term	A term that is part of Planguage.	Structuring specifications.	Glossary contains a set of Planguage terms.
User-Defined Term	A term defined by users.	Identifying 'local' user terms.	It should be short and descriptive.
Type:	Type or category of a term.	Declaring the Planguage category of a user-defined term.	Type can be implicitly or explicitly declared.
Tag:	An identifier for a Planguage term or a user-defined term.	Providing a unique 'local' reference to a term.	Hierarchical tags can be used. These can be used in full (very explanatory) or abbreviated depending on context.
Gist:	A rough, informal, brief description or summary.	Getting consensus initially. Summarizing finally.	Usually not a precise, detailed or complete definition. For a scalar parameter, 'Ambition' can be used to express the ambition level.
Description:	A description.	Explaining terms.	Level of explanation detail should match the context.
Definition:	A definition, usually expressing the relationship to other user-defined terms.	Defining terms.	Synonyms are 'Defined' and 'Defined As.'
Version:	A date stamp. A time stamp can optionally be added.	Identifying a specific instance of a specification.	For example: 'Version: October 7, 2004 10:20.'
Stakeholder:	Any person or organizational group with an interest in, or ability to affect, the system or its environment.	Understanding who has to be consulted or considered when specifying requirements.	Usually a set of several different stakeholders is identified.
Authority:	The stakeholder or role responsible for determining status and enforcing use.	Identifying where the power resides.	The authority has the power to determine and change a specification. Also to control its availability.
Owner:	The stakeholder or role responsible for the overall specification itself.	Identifying the specification owner.	The owner usually is responsible for the updating of a specification.

Continued

Table 1.1 Continued

Basic Planguage Parameters, Concepts and Icons

Concept or Parameter	*Meaning*	*Used for*	*Note also*
Readership:	The stakeholder(s) or role(s) who will or might use the specification.	Identifying the specification user(s) or audience level to communicate to.	A synonym is 'Intended Readership.' A parameter such as 'Specification User' or 'Process User' could be used instead.
Status:	The approval level of the specification.	Identifying which version of the specification is being used.	For example: 'Status: Draft.' See glossary for additional terms to express approval level.
Quality Level:	The quality level of the specification in relation to its rules.	Stating the estimated defect density in a specification.	For example: 'Quality Level: 3 remaining major defects/page.'
Qualifier: [...]	A qualifier adds more specific detail to the specification regarding time, place and event conditions, [when, where, if].	States the conditions applying to a specification for it to be valid: the [time, place, event] conditions.	The keyed icon for Qualifier is '[ ]' as in '[Qualifier Condition 1, Qualifier Condition 2, ... Qualifier Condition n].' The '[...]' icon is used far more than the parameter, Qualifier.
Source: <-	Where exactly a given specification or part of it, originated.	Used to enable readers to quickly and accurately check specifications at their origin.	The icon for source is '<-'. Usually the icon is used in specifications, rather than the term 'Source'.
Assumption:	Any assumption that should be checked to see if it is still applies and/or is still correct.	Risk Analysis	Other more precise parameters should be used if possible, for example, Dependency, Risk.
Note: "..."	Any additional comments or notes, which are relevant.	Used to provide additional information likely to help readers.	Any notes are only commentary and are not critical to a specification. 'Comment:' could be used as an alternative.
Fuzzy <...>	Identifies a term as currently defective and in need of improvement	Alerting the reader and author that the term is not trustworthy yet or lacks detail.	The keyed icon for fuzzy is '<imprecise word>'. The '<>' icon is always used.
Set Parentheses {...}	Identifies a group of terms, linked in some way, forming a set or a list.	Explicitly shows that a set of terms is being specified.	The context explains why the terms are a set. Usually, all terms are of the same Type.

1.4 Rules: Generic Rules for Technical and Management Specification

Here are some very basic generic rules, for *any* type of specification. You will find that in spite of their 'obviousness' and simplicity, they are quite powerful. Most of my clients use some variation of these 'by choice'.

Tag: Rules.GS.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Note: These rules are rather lengthy, as additional explanatory text is present. Readers should abbreviate as appropriate.

R1:[3] **Tag**: Specifications must each have a unique identification tag.

R2: **Version**: Specifications must each have a unique version identifier. By default, use the date (and maybe also, time), as the version identifier.

EXAMPLE Version: October 7, 2004 09:00.

R3: **Unique**: Specifications shall exist as *one official 'master' version only*. Then they shall be re-used, by cross-referencing, using their identity tag. Duplication ('copy and paste') should be strongly discouraged.

R4: **Owner**: The person or group responsible for authorizing a specification should be stated ('Authority' would be an alternative or supplementary parameter, though it is a different concept!).

R5: **Status**: The status for using a specification should be given.

EXAMPLE Status: SQC Exited.

R6: **Quality Level**: All specifications shall explicitly indicate their current quality level, preferably in terms of the measure of 'number of remaining major defects/page' against the relevant official standard which applies.

[3] The number is a rule tag (or identification, if you like) and the word after the colon is an equivalent alternative tag for referencing the rule. The following references are possible Rules.GS.R1, Rules.GS.Tag, Standards.Rules.GS.Tag and other combinations. The dot indicates that what follows is part of a set of things named by the term preceding the dot. For example, GS is part of a set of things called Rules.

EXAMPLE Quality Level: Less than 1 remaining major defect/page.

EXAMPLE Quality Level: Undetermined.

R7: **Gist**: Where appropriate, specifications should be briefly summarized by a Gist statement. For performance requirements, 'Ambition' is a preferred alternative.

R8: **Type**: The type of every concept within specifications should be clear. It should be explicitly specified after every new parameter tag declaration unless the type will be immediately obvious to the intended readership.

EXAMPLE *ABC1*: Type: Function.

R9: **Clear**: Specifications should be 'clear enough to test' and 'unambiguous to their intended readers.'

R10: **Simple**: Complex specifications should be decomposed into a set of elementary, tagged specifications.

R11: **Fuzzy**: When any element of a specification is unclear then it shall be marked, for later clarification, by <fuzzy angle brackets>.

R12: **Comment**: Any text which is secondary to a specification, and where no defect in it could result in a costly problem later, must be clearly identified. It can be written in *italic text* statements, or headed by suitable warning (such as Note, Rationale or Comment), or written in "quotes," and/or moved to footnotes. Non-commentary specification shall be in plain text. *Italic* can be used for emphasis of single terms in non-commentary statements. Readers should be able visually, at a glance without decoding the contents, to distinguish between 'critical' and *'non-critical'* specification.

R13: **Source**: Specification statements shall contain information about their source of origin. Use the '<-' icon and state the source person and the date, or the source document with detailed statement reference.

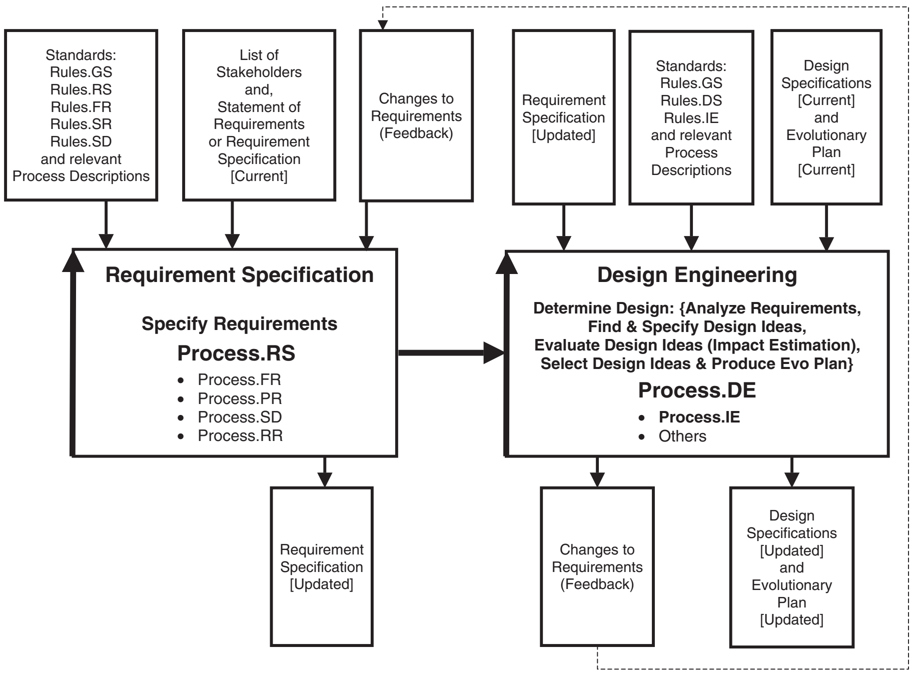
R14: **Assumptions**: All known assumptions (and any relevant source(s) of any assumptions) should be explicitly stated.

The 'Assumption' Planguage parameter can be used for this purpose. But there are also a number of alternative ways, such as {Risk, Source, Impacts, Depends On, Comment, Authority, [Qualifiers], If}. In fact, any reasonable device, suitable for the purpose, will do.

R15: **Risks**: You must specify any factors, which constitute known or potential risks. You must identify risks explicitly.

There are a wide variety of devices for doing so, including the explicit Planguage statement: 'Risks.'

EXAMPLE Goal [Market Y]: 60%.
Risks: Market Y will have more competition than now.



Standards: Rules.GS Rules.RS Rules.FR Rules.SR Rules.SD and relevant Process Descriptions
List of Stakeholders and, Statement of Requirements or Requirement Specification [Current]
Changes to Requirements (Feedback)
Requirement Specification [Updated]
Standards: Rules.GS Rules.DS Rules.IE and relevant Process Descriptions
Design Specifications [Current] and Evolutionary Plan [Current]
Requirement Specification
Specify Requirements
Process.RS
• Process.FR
• Process.PR
• Process.SD
• Process.RR
Design Engineering
Determine Design: {Analyze Requirements, Find & Specify Design Ideas, Evaluate Design Ideas (Impact Estimation), Select Design Ideas & Produce Evo Plan}
Process.DE
• Process.IE
• Others
Requirement Specification [Updated]
Changes to Requirements (Feedback)
Design Specifications [Updated] and Evolutionary Plan [Updated]

Notes:
Iteration of the processes has been allowed for by including existing specifications as potential inputs. Qualifying square brackets have been used around descriptive words, which are added to assist understanding. The aim is to show how the rules and process descriptions discussed in this book fit together. This diagram shows procedure steps P1 and P2 of the Generic Project process (Process.GP). These same processes are used during Manage Evolutionary Project (Process.GP.P3) – that is during Evolutionary Project Management – in order to update the requirements, the ideas and the Evo plan (see Figure 1.6).

The abbreviations used in this figure (and in the rest of the CE book) are as follows:

GP	Generic Project	RR	Resource Requirements
GS	Generic Specification	DS	Design Specification
RS	Requirement Specification	DE	Design Engineering
FR	Function Requirements	IE	Impact Estimation
SR	Scalar Requirements	EVO	Evolutionary Project Management
PR	Performance Requirements	SM	Strategic Management
SD	Scale Definition	DC	Delivery Cycle

Figure 1.5
An overview of the Planguage-defined requirement and design processes.

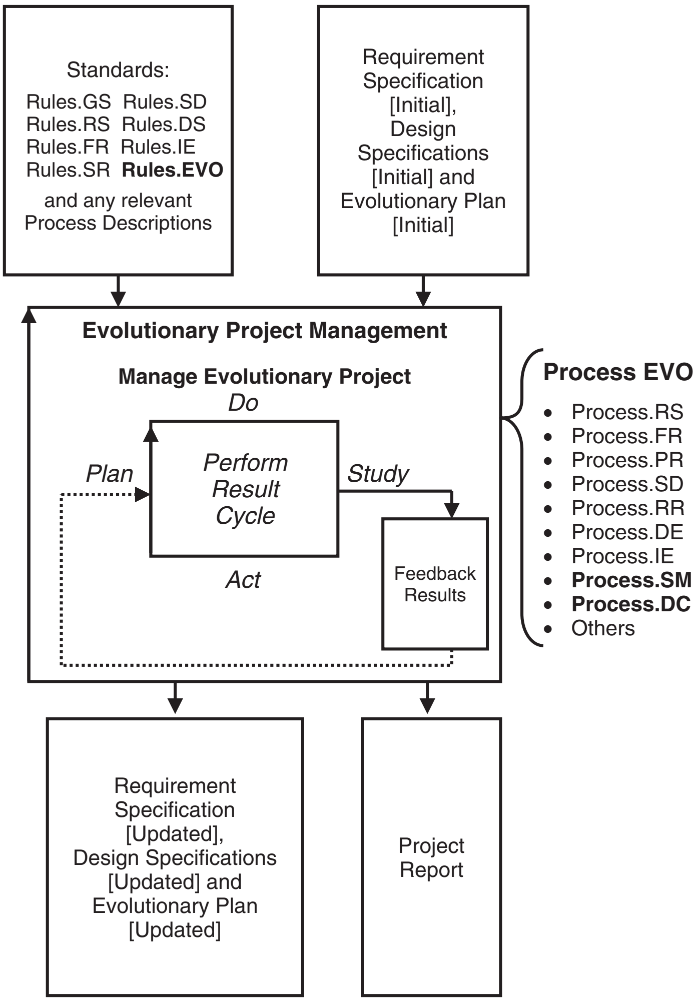

Standards:
Rules.GS Rules.SD
Rules.RS Rules.DS
Rules.FR Rules.IE
Rules.SR **Rules.EVO**
and any relevant Process Descriptions
Requirement Specification [Initial], Design Specifications [Initial] and Evolutionary Plan [Initial]
Evolutionary Project Management
Manage Evolutionary Project
Do
Plan
Perform Result Cycle
Study
Act
Feedback Results
Process EVO
Process.RS
Process.FR
Process.PR
Process.SD
Process.RR
Process.DE
Process.IE
Process.SM
Process.DC
Others
Requirement Specification [Updated], Design Specifications [Updated] and Evolutionary Plan [Updated]
Project Report

Figure 1.6
An overview of the defined Planguage process, which supports Evolutionary Project Management, Process.GP.P3 or in more detail, Process.EVO in Chapter 10.

1.5 Process Description: Generic Project

Process: Generic Project

Tag: Process.GP.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: A process specification giving an overview of the entire Planguage process for a project.

Entry Conditions

E1: The **Generic Entry Conditions** apply (*see separate specification for Generic Entry Conditions below*).

The *raw* requirements should have been gathered. The known sources of requirements should be identified and listed. These include:

- all the critical stakeholders
- all the currently identified requirements with detailed sources (use '<-' and, state who or which document) and any justification for these requirements (use the Rationale parameter).

Procedure

P1: **Specify Requirements** [**Initial**]: Specify the initial top-level requirements (*see Chapters 2, 3, 4, 5 and 6 as appropriate*).

P2: **Determine Design** [**Initial**]:

P2.1: **Analyze the Requirement**: Consider the stakeholder value and the delivery order for the requirements. Identify any constraints and any conflicts. Establish the scope for the system design.

P2.2: **Find and Specify Design Ideas**: Identify and specify the initial top-level design ideas to meet the requirements (*see Chapter 7*).

P2.3: **Evaluate Design Ideas**: Estimate the impacts of all the design ideas on all the requirements (*see Chapters 7 and 9*).

Re-do P1 to P2.3, until a reasonable balance between requirements and costs is obtained.

P2.4: **Select Design Ideas and Produce Evo Plan**: Produce an initial overview, long-term evolutionary plan of the sequence of Evo steps. That is, a plan for starting early delivery of required results by implementing the design ideas in a series of small result cycles. Each result cycle using, say 2% of total project time. (*That is, each result cycle is an Evo step. Note, an Evo step contains one or more design ideas.*)

Determine the sequence of step delivery of the potential Evo steps. Do this by calculating for each potential step, the performance to cost ratio, or ideally you would use the 'stakeholder view' of the value to cost ratio (the value being the benefits the stakeholders consider they will obtain from the system improvements). Ideally, sequencing should be in order of descending ratios, but consideration needs to be given to any associated dependencies (*see Chapters 7 and 10*). *Note this plan will be modified, within the result cycles, using the feedback provided by the results of implementing the design ideas (see below).*

P3: **Manage Evolutionary Project**: Iterate Plan-Do-Study-Act (PDSA) evolutionary result cycles until the exit conditions (below) are met. *Each result cycle implements the next Evo step and provides feedback to modify the design, and maybe, to adjust requirements to more realistic levels (within each result cycle, the processes Specify Requirements and Determine Design are reiterated to carry out any more detailed work required as part of the implementation of the Evo step, and to cater for any changes required as a result of the feedback), (see Chapter 10, 'Evolutionary Project Management').*

Note: When using Evo, as long as the Evo result cycles are delivering to the planned levels, the need for initial management review is considerably decreased (if not eliminated) as the resource commitment for each delivery step is only about 2% of the project total.

Exit Conditions

X1: The **Generic Exit Conditions** apply (see separate specification for Generic Exit Conditions below).

X2: Cease doing Evo steps (P3) when either the stakeholder requirements are met, or resource budgets are exhausted. In other words, stop when the performance requirements are met at planned levels, or when resources (budgets) are 'used up' at their planned levels.

Generic Entry and Exit Process and Conditions

Here is a process that can be used as a generic entry process or a generic exit process. The benefit of having one master generic process is that it is easier to review and update.

Process: Generic Entry or Generic Exit

Tag: Process.GE.E or Process.GE.X.

Version: October 7, 2004.

Owner: Systems Engineering Process Owner.

Status: Draft.

Gist: A generic process description that applies by default to all entry and exit processes.

Procedure

P1: Check all the conditions that apply.

P2: Note which conditions cannot be met.

P3: Decide if we can or must ignore specific failed conditions (waver).

P4: Attempt to correct, or help others to correct, any failed conditions in need of correction.

P5: Report status of the process in writing.

P6: Help management understand the reasons for and the risks of ignoring the problem of any failed or waved conditions.

P7: If management insists on overriding your advice, make sure the responsible manager, after being informed of the risks, is documented as overriding the formal process intentionally. (Make sure we know who to blame later and then they take the responsibility.)

P8: For exit only: Ensure any process improvement suggestions have been submitted to the relevant process owners.

P9: Allow exit/entry when all conditions are either met or waived.

Note: To simplify matters, no entry or exit conditions have been specified for this process!

Generic Entry Conditions

Scope: For systems engineering, all specification entry processes.

Owner: Systems Engineering Process Owner.

User: Specification Author [Default User: SQC Team Leader].

E1: All logically necessary input information for complete and correct specification is available to the specification author. This includes up-to-date documentation regarding specification standards.

E2: All input documents have successfully exited from their own quality control process.

Note: This usually implies between 0.2 and 1 maximum remaining major defect(s)/page (A page is 300 words of non-commentary text). 'Remaining major defects' is explained in Chapter 8, 'Specification Quality Control.'

E3: The specification author is adequately trained or, assisted by a qualified person.

E4: The specification author agrees that they are ready to successfully carry out the specification work.

E5: There is appropriate approval, including funding, for the specification process to proceed.

Generic Exit Conditions

Scope: For systems engineering, all specification exit processes.

Owner: Systems Engineering Process Owner.

User: Specification Author [Default User: SQC Team Leader].

X1: The specification author claims to have followed the specified process description standard.

X2: The specification author claims to have followed all generic and specific rules, which apply.

X3: Relevant SQC has been carried out and the quality level of each output specification meets its stated SQC criteria. By default, the quality level for any specification is that no more than 0.2 major defects/page[4] may remain. (A page is 300 words of non-commentary text.)

Note, for some processes, there will be an explicit statement on SQC criteria, which overrides this generic exit condition.

X4: As an additional optional measure, a cursory check of the specification by the author's supervisor shows that there is reasonable compliance with applicable rules. In practice, no major defects should be found when a relevant sample (size and content) of the specification is SQC checked for 15 minutes.

X5: Any process improvement suggestions identified have been submitted to the relevant process owners.

1.6 Principles: Generic Project

Principles are *teachings*, which you can use as guides to sensible action. Here is a set of fundamental principles:

1. **The Principle of 'Controlling Risk'**
   There is lots of uncertainty and risk of deviation from plans in any project.
   You cannot eliminate risk. But, you can document it, plan and design for it, accept it, measure it and reduce it to acceptable levels.
   You may want to avoid risk, but it doesn't want to avoid you.

2. **The Principle of 'Storage of Wisdom'**
   If your people are not *all* experienced or geniuses,
   You need to store *their* hard-earned wisdom in *your* defined process.

[4] A maximum of 0.2 remaining major defects/page is a very high standard. Beginners should try for about 2.0 and work towards better levels.

Capture wisdom for reuse,
Fail to write it, that's abuse!

3. **The Principle of 'Experienced Geniuses'**
   If you *do* have any experienced geniuses, don't just let *them* save projects;
   They should share *their* wisdom with colleagues, on how to avoid failures.
   Those who learn the hard way,
   Should share their easy way.
4. **The Principle of 'Grass Roots Experience'**
   Your grass roots people will *know* what is wrong with your work standards,
   So let *them* suggest improvements, every day.
   The soldier who has the boot on knows where it pinches.
5. **The Principle of 'Short and Sweet'**
   Keep your standards short and sweet,
   A single page will do the feat.
   Brevity is the soul of wit,
   All essentials, a page do fit.
6. **The Principle of 'Don't Refuse to Reuse'**
   Reuse good specifications, and don't repeat them,
   Once said suffices, no repetitious vices.
   Write once, use many.
7. **The Principle of 'High Standards'**
   Have high standards for your work process entry, to save yourself grief,
   Have high standards for your work process exit, to your friends' great relief.
   Note work standard conditions for success,
   Respect them; even in duress.
8. **The Principle of 'Quality In, From the Beginning'**
   Quality needs to be designed into processes and products,
   Cleaning up bad work is a loser, but cleaning early is better than late.
   A stitch in time still saves nine,
   But an ounce of prevention is still worth a pound of cure.
9. **The Principle of 'No Simpler'**
   The optimum guidance lies somewhere between anarchy,
   And too much bureaucracy.
   Things should be as simple as possible,
   But no simpler.[5]

[5] "Physics/theories/things should be as simple as possible, but no simpler". Reputed quote of Albert Einstein. Nobody seems able to prove he actually said it, but it is acknowledged to be in his spirit. Calaprice, Alice [Editor]. 2000. *The Expanded Quotable Einstein.* Princeton University Press. ISBN 0-691-07021-0.

10. **The Principle of 'Intelligent Insubordination'**
    A work process 'standard' isn't a law, just good advice,
    Ignore it, if you've better 'words from the wise'.
    Rules were made to be broken wisely.

1.7 Additional Ideas

Continuous Process Improvement

Conventional ways of getting control over systems engineering projects include:

- **resource** allocation adjustment (time, people, talent, money, sponsorship)
- **ambition** level adjustment (performance to fit within budgets)
- shift of **responsibility** (outsourcing, purchasing, contracting, democratization)
- **priority** management (sacrificing some things to get others, tradeoffs).

There is a less-understood *addition* to these ideas: **process control**. It is to get control over results by getting control over the *work processes* producing the results. In concept, this is Statistical Process Control with its famous Plan-Do-Study-Act (PDSA) cycle as taught by Shewhart, Deming and Juran (Deming 1986).

'Process control' is sufficiently well known within manufacturing. However, surprisingly, it has not become conventional practice within systems engineering. There are two main areas where its use is lacking.

First, process control is rarely exploited in the area of *project management*. This is in spite of there being 'software' literature, which documents good experience with process control since the

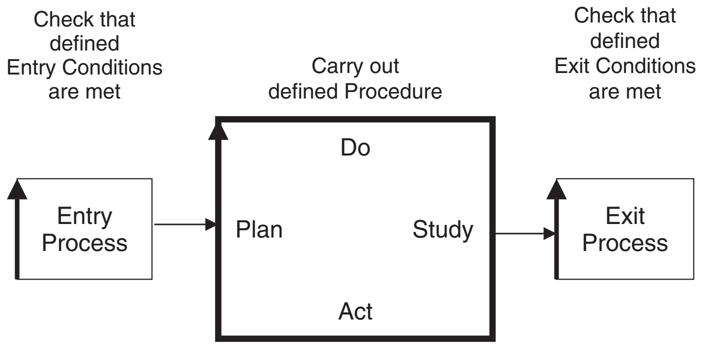

Check that defined Entry Conditions are met
Carry out defined Procedure
Check that defined Exit Conditions are met
Entry Process
Do
Plan
Study
Act
Exit Process

Figure 1.7
A *simplified* PDSA process cycle diagram as a basis for work process control, consisting of an entry process, a procedure and an exit process.

1970s: for example by Harlan Mills at IBM (Mills 1980), references such as (Gilb 1988) and, even military IT standards within the USA (such as MIL-STD-498 in 1994) (see Larman and Basili, 2003, for historical overview.) The problem is that this 'software' documentation is little known, having simply not been adequately recognized in mainstream *project management* literature. (In fact, there appears to be almost no reference at all to evolutionary project delivery and process control. The Waterfall method unfortunately dominates, according to my informal bookshop surveys and speaking with professional project management people.)

Secondly, the PDSA cycle concept is also underutilized in systematic *process improvement*. Use of numeric feedback for control is often not understood and not practiced. This is, however, being addressed in emerging standards for systems engineering and in US DoD 'Mandatory Guidelines' (DoD Evolutionary Acquisition 1998).

The key concept is that if a well-defined process is followed, then the process output performance levels will be a consistent and predictable result of that process. If attempts are then made to *change* the process, we can assume that systematically changed performance results (hopefully, better levels and lower variability) can safely be attributed to the process change, not chance. Unfortunately this powerful concept is frequently ignored. The false dogma is often spread that defined repeatable processes lead to quality. (In fact, this is only the initial stage of achieving a stable process, which is then ready for process change, as the prior stability enables proof of the cause-and-effect of the change.)

It is important that work process **standards** be the vehicle for *continuous, systematic work process practice improvement (productivity improvement)*. They must not remain static, when there is better know-how. They must not stand in the way of improvement. They must lead the way and teach the way. They must be *easily changed* and *frequently changed* to incorporate better ideas quickly, and *easily adapted* to suit changing circumstances or tailoring for local circumstances. The actual usage of work process standards must be measured, motivated and taught by using Specification Quality Control (SQC) sampling. SQC measures specification conformance to two classes of work process standards: official rules and exit/entry conditions.

Normally no more than one significant deviation (one major defect/page) from the specification rules should be allowed. Yet without SQC, 100 or more major defects/page will be your fate. This may seem astounding to people who have not measured it, but this, in my experience, is the norm in most organizations throughout the world.

Why Process Control?

- Use of Best Practice
- Reuse of Ideas
- Predictable Output from Stable Process
- Rapid Dissemination of Changes
- Ability to detect any 'Bad' Process Changes
- Process Measurement and Benchmarks

Not just 'having a Process', but using it as 'a vehicle for Change'.

Continuous work process improvement for a large organization can involve making changes to company standards and practices at the rate of 1,000 ideas implemented per year – as documented at IBM, Research Triangle Park (Mays 1995) and IBM Rochester (Minnesota) Laboratories (IBMSJ 1994). This process change rate seems to result in annual productivity increases of about 40%, as recorded for example at Raytheon Defense Electronics over several years (Dion 1993; Haley et al. 1995; see also Section 1.8 below; over the years studied, a total productivity increase of 270% was reported).

EXAMPLE **Calculating the effect of detected defects, if uncorrected, on the timescales of a project**
At a major U.S. multinational in October 1999, eight managers did a sample SQC on an 82-page system requirement specification. The only rules used were, 'clear, unambiguous, no design specifications in the requirements.' They found about 60 major defects/page.
Assumptions: My SQC experience has determined that:

- only about one third of the defects that are really there will be found by staff inexperienced in using SQC at the first pass
- each defect will result in 'an order of magnitude' extra work to fix when found downstream.

It is also assumed that there are 200 days per year and 8 hours per work-day (1600 hours/year).
Using these assumptions, it can be calculated that the project will incur 82 (number of pages) × 60 (number of defects/page), × 3 (as only a third effective in finding defects), × 10 (number of hours/defect) additional hours correcting defects = 147,600 hours or approximately 92 person years.
We can assume the probability of a major defect actually resulting in an average 10 hour delay is about 25%–35%. So at 25% we would lose 36,900 hours.
For a project with 10 programming staff, this meant roughly two years' delay.
We later that afternoon were told that the project using this 'approved' specification was actually at least one year late, probably 2 years late. This had in fact been predicted fairly accurately by our analysis, before we were told the reality!
In such an environment, simply continuing to fix specification defects as they are detected is not the sensible option. *Continuous process improvement* needs to be used to drive down the number of defects being injected into the specifications.

See also Chapter 8 and the Glossary for further detail on Specification Quality Control (SQC) and the Defect Prevention Process (DPP).

1.8 Further Example/Case Study: Continuous Process Improvement at Raytheon

This Raytheon case study outlines how measurable process improvement can be brought about using Specification Quality Control (SQC) and Continuous Process Improvement. Within Raytheon's Equipment Division, software process improvements have yielded:

- a 7.70 US dollars return on every dollar invested
- a greater than two-fold (2.7×) increase in productivity
- as measured by the Software Engineering Institute (SEI) Capability Maturity Model (CMM), an evolution from Level 1 (Initial) through Level 2 (Repeatable) to Level 3 (Defined) process maturity (and later beyond that).

More detail can be found in Raymond Dion's account of the software process changes within Raytheon (Dion 1993; Haley et al. 1995).

Background

Raytheon, a diversified, international, technology-based company, is one of the 100 largest corporations in the US. The Equipment Division is one of eight divisions, and 11 major operating subsidiaries within Raytheon, with annual sales that comprise about 13% of the corporation's $9.1 billion annual sales. In early 1988, many Software Systems Laboratory (SSL) projects were delivered late and over budget. That year, the SSL rated itself at CMM Level 1 (Initial), using the SEI capability-assessment questionnaire.

Aim

As a result, in mid-1988, the Equipment Division started a process-improvement initiative within the SSL. Within the initiative, four working groups directed the major activities: Policy and Procedures, Training, Tools and Methods, and Process Database (metrics). The initiative's fundamental aim was the continuous improvement of the development and management process. Their strategy was to use a three-phase cycle of stabilization, control and change in accordance with the (PDSA) principles of W. Edwards Deming and Joseph Juran.

Financing the Improvements

Discretionary funding (overhead, independent research and development, and reinvested profit) was the chosen solution. However, in order to convince management to persevere, this approach required two important ingredients. First, there had to be some *short-term benefit to ongoing projects* and, second, there had to be a *meaningful quantification of what the benefit was.*

Measuring the Effects

In launching the initiative, they had to consider how individual small improvements, implemented more or less in parallel, would interact to produce a net loss or gain. They decided it would be easier to measure the overall effect of change on the 'bottom line'.

Calculating Savings

Raytheon used Philip Crosby's approach (Crosby 1996) to analyze a database of 15 projects. The analysis, indicated that they had eliminated about $15.8 million in *rework costs* through the end of 1992 (four and a half years). (Hewlett Packard, using this author's SQC methods, reported similar results (Grady and Van Slack 1994)). The Raytheon appraisal costs (a term meaning cost of auditing, testing, reviews and inspections) had increased by 5%. The increased rigor with which they conducted design and code inspections (SQC), accounts for some of this increase, but most of the Raytheon result is due to a 30% *decrease* in total project cost, which has pushed up appraisal cost *proportionally.*

Early delivery of one Raytheon project was reported to have given them a $10 million bonus from their customer. It was considered entirely due to the initiative. There were several other tangible benefits from the initiative. The saving in rework costs was only *one* of them.

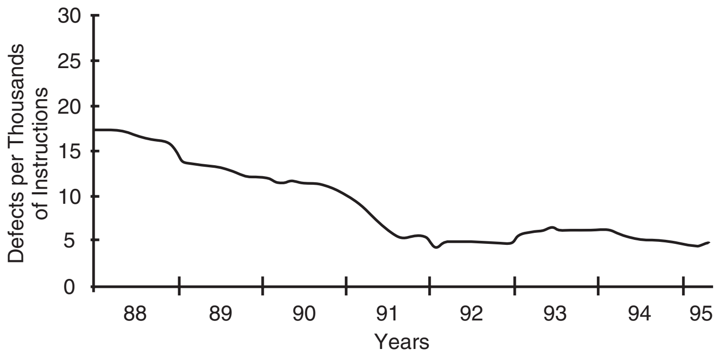


Figure 1.8
Another benefit from the effort: overall product quality, measured by software defect density, improved by about 3 to 1, from 1988 to 1995 at Raytheon (Haley et al. 1995).

1.9 Diagrams/Icons

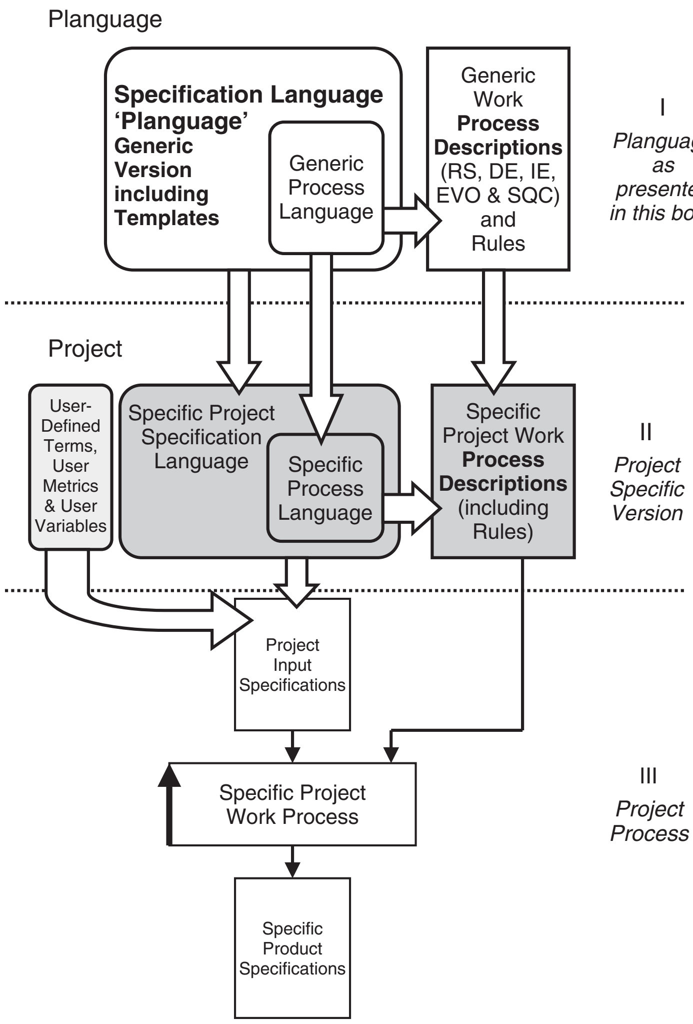

Planguage
Specification Language 'Planguage' Generic Version including Templates
Generic Process Language
Generic Work Process Descriptions (RS, DE, IE, EVO & SQC) and Rules
I
Planguage as presented in this book
Project
User-Defined Terms, User Metrics & User Variables
Specific Project Specification Language
Specific Process Language
Specific Project Work Process Descriptions (including Rules)
II
Project Specific Version
Project Input Specifications
Specific Project Work Process
III
Project Process
Specific Product Specifications

Figure 1.9

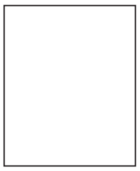
I. At the top of the diagram, the two main, generic components of Planguage, the specification language and the process descriptions are shown. (These two components correspond to the version of Planguage presented in this book.)

II. In the middle of the diagram, the specific version of Planguage (the project specification language and project process descriptions) selected for use by a project is shown. This specific version will have been tailored by the project. In addition, a project will have user-defined data. The user-defined data will always be unique to a project. It comprises the user-defined terms, actual numeric values (user metrics) and any user-assigned, non-numeric variables of the project specifications.

III. The bottom of the diagram is a generic model of a project process. It shows how the various components of the project specific version of Planguage (and the product data) map onto the project process.

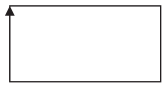
Some Basic Planguage Icons

Document or Specification



A rectangle

Process

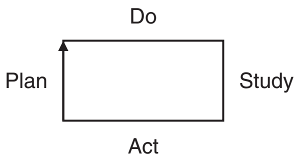


A rectangle with an upwards pointing
arrow on its left hand side.
The arrow reminds us of the cyclical nature of processes.

Plan-Do-Study-Act Process Cycle

The four sides of the process icon symbolically represent the Shewhart process-cycle definition of 'Plan-Do-Study-Act'.

The process **input/output axis** is vertical and the process **control axis** is horizontal.[6] Specifications and other input materials are diagrammed as entering from the north and exiting from the south. Previous processes are connected from the west and subsequent processes are entered from the east. These conventions are independent of the PDSA activities, since one can enter and exit to and from any of these four process task types. (That is, you can step on and off the cycle at any point.)



[6] The traditional view as shown by Deming is a circle form with four arrows. I have chosen the rectangle as it is easier to generate and has other nice properties. I hasten to point out that Deming taught that it did not matter where in the cycle one entered a PDSA process, nor where one exited, though he was not so concerned with exit, as he viewed the cycle as an eternal process-control cycle, as long as there were competitive pressures to improve things. I believe this is true and, so, I hope my choice of representation does not inhibit the reader from entering and exiting processes wherever convenient or realistic (P, D, S or A).

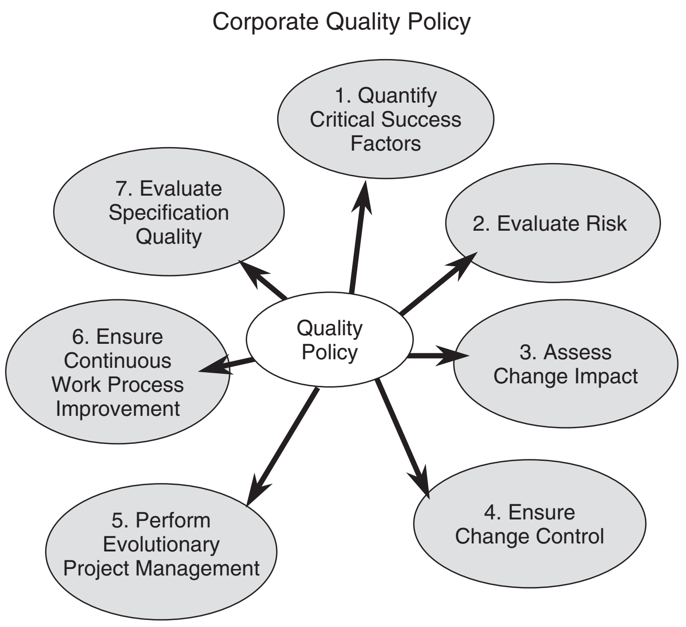

Corporate Quality Policy
Quality Policy
1. Quantify Critical Success Factors
2. Evaluate Risk
3. Assess Change Impact
4. Ensure Change Control
5. Perform Evolutionary Project Management
6. Ensure Continuous Work Process Improvement
7. Evaluate Specification Quality

Figure 1.10
Example of a corporate policy standard.

Notes Supporting the Example of a Corporate Policy Standard

1. **Quantify Critical Success Factors**:
   All critical success factors (function, performance and resource) for any activity (planning, systems engineering and management) shall be expressed clearly, unambiguously, measurably and testably at all stages of consideration: presentation, evaluation, construction and validation.
2. **Evaluate Risk**:
   In any planning or systems engineering work we shall explicitly document all notion of suspected or possible elements of risk or uncertainty, so nobody reading it can be in the least doubt as to the state of our certainty and knowledge.
3. **Assess Change Impact – To Exercise Control over Multiple Dimensions of Performance and Budget**:
   All design ideas (strategies, system components, processes or other devices) shall be evaluated with regard to their effects on all the critical objectives and budgets. Initially, this should be by estimates, which are based on facts and experience. On delivery, the design ideas shall then be evaluated by actual measurements taken as early and as frequently as possible.

4. **Ensure Change Control – Configuration Management and Traceability**:
   All statements of objectives, budgets, design ideas, and estimates and measures of the impact of design ideas on objectives and budgets shall be captured with explicit detailed information as to their sources, so that detailed change control is made effective and efficient.
5. **Perform Evolutionary Project Management**:
   All projects whether concerning organizational issues or product development, shall be controlled by a Plan-Do-Study-Act process control cycle. They shall have small increments of cost and time (in the 2% to 5% range normally) before attempting to deliver useful customer increments of function and/or performance improvement (at least some sort of field trial). Where there is any choice of incremental step content we shall choose the increment which gives the greatest quantified impacts in total on all critical customer or project objectives, with least resource expenditure.
6. **Ensure Continuous Work Process Improvement**:
   Practical priority will be given to measurable continuous improvement of all work processes in systems engineering, management and other company activities. Plans for type and degree of improvement will be budgeted; and progress towards improvement objectives will be measured. The ambition level will be world-class levels and to be the leader in any area. As a practical matter all employees are expected to participate in analysis of current defects found by quality control (for example, specification quality control (SQC) and test) and to spend effort improving the current work environment to eliminate 50% of the current defects every year over the next few years.
7. **Evaluate Specification Quality**:
   All documents, capable of producing a significant impact on our performance levels, must be evaluated using the best available quality control process. These documents must meet an appropriately high quality standard (that is a low numeric value for the 'maximum possible remaining major defects/page' as specified in our written standards and policies) before being released to any internal or external customer for serious use. The ultimate release level shall be state of the art (between 0.3 and 3.0 remaining major defects/page).

1.10 Summary: Planguage Basics and Process Control

This chapter has provided an introduction to Planguage and, hopefully, set the rest of the book in context. The main Planguage concepts introduced in this chapter have been *processes* and *continuous process improvement* through use of process *standards*. Many examples of process standards will be found throughout this book. They aim to provide practical, step-by-step advice on how to implement Planguage.

Planguage is not a prescription of how I feel you should do things. It is a framework for you to discover how you best can do things yourself. Planguage is *open for change* from any source, at any time, for any good reason. It is intended to be totally in tune with the need for *continuous improvement* of all competitive systems and processes.

If Planguage doesn't save time and effort and improve quality, it fails. Don't use it! Please do not misunderstand Planguage as if it is an 'imposition of a lot of bureaucratic detail.' I hate bureaucracy as much as you do! But I hate failure even more. So, I am willing to use the Planguage disciplines; I find that they pay off and make my professional life easier and more successful. (Note: The Planguage methods actually work in *most* problem-solving situations; they can even be used in your *personal* life too!)

Planguage is concerned with *getting control over things*. If you want to be more in control of your work, Planguage has many practical techniques to help you. It takes some *learning*. It takes some *work to implement*. It takes *time to change the culture* around you.

In fact, human culture changes can be frustratingly slow; they can take years! But if you don't start this evolutionary process *now*, *this* week, *this* project, then the problems will get worse, not better. Can you afford to ignore the evidence from several major corporations, such as Raytheon, that continuously improved best practice standards can lead to substantial improvements in your team productivity annually over the next few years?

Chapter
2

Introduction to Requirements

Why?

Glossary Concepts

- System
- Stakeholder
- Requirement
- Attribute
- Vision
- Function
- Performance
- Objective
- Quality
- Resource Saving
- Workload Capacity
- Resource
- Cost
- Budget
- Design Idea
- Condition
- Target
- Constraint
- Benchmark

2.1 Introduction to Requirements Specification

Peter Morris, having studied numerous projects in the US and UK covering the period from 1940 to 1990, identifies that one of the major causes of project problems is that our current management and engineering culture consistently fails sufficiently to *articulate* requirements or *cope with change* in them (Morris 1994).

More recently, in 2001, having conducted a thorough review of the recent systems engineering industry literature, Ralph Young concludes that the causes of project failure are ineffective practices for handling requirements. He estimates the necessary improvements in such practices could be financed by approximately one third of the current total cost of project failures. Additional gains would be that customer satisfaction and the quality of results would also improve (Young 2001).

You probably feel that you need to ask more probing questions about the project requirements that *you* are working on. You are likely, unfortunately, to be able safely to assume that nobody in your senior management and none of your customers has a well-developed sense of *exactly* what requirements they really want or need. They may all have the dangerous *illusion* that they do. However, they are unlikely to have a clear enough requirement specification. Nor are they likely to have requirement ideas which are 'shared precisely' by all their colleagues and the other stakeholders.

Definition of Requirements

Requirements give information to the system designers and to a wide range of stakeholders. They state what the stakeholders want the system to achieve.

Requirements can be classified into 'requirement types' as follows:

0. **Vision**: at the highest level, the future direction for a system.
1. **Function Requirements**: *what* a system has to 'do': the essence of a system, its mission and fundamental functionality.
2. **Performance Requirements**: the performance levels that the stakeholders want – their objectives. *How good?* These can be further classified as:

   - **Qualities**: *how well* the system performs, for example: usability, availability and customer satisfaction.
   - **Resource Savings**: the *required improvement in resource utilization*: relative economic and other resource savings compared to defined benchmarks. These are known simply as 'Savings.'

- **Workload Capacities**: *how much* the system performs. In other words, the *required capacity of the system processes*. For example, system peak processing volumes, speeds of execution and data storage capacity.

3. **Resource Requirements**: the *levels of resources* that stakeholders plan to expend to develop and operate a system. Resources have to be balanced against the stakeholders' perceived values gained from the system functions and the system performance levels.
4. **Design Constraints**: these are any design ideas that must be included in the system design.
5. **Condition Constraints**: these are any additional constraints to those imposed by the function requirements, the performance requirements, the resource requirements and the design constraints. Condition constraints are often used to capture system-level constraints (for example, 'the system must be legal in Europe').

From the viewpoint of understanding 'competitiveness', 'levels of achievement' and 'associated risk,' the **performance requirements** are by far the most interesting requirements. Yet, traditionally, too much attention has been given to specification of function requirements and resource requirements (such as financial budgets, deadlines and headcounts). We need a more balanced requirement specification that includes all targets and all constraints. They all need to be equally clear and equally capable of being tested.

Key Issues for Requirements

Here are some key issues to consider when using or specifying requirements:

Identifying the critical stakeholders

Failing to identify the critical set of *stakeholders* is a common problem. The stakeholders for a system are anyone affected by the system or who can impact the system. This includes system users, maintainers, financiers, managers, developers, critics and others. If you fail to consult and analyze the critical stakeholders, then your requirements will risk being dangerously incomplete. By definition this will threaten the existence of your system, or at least its profitability.

Hint: Consider the entire lifecycle (including retirement or replacement) of a system or product when looking for stakeholders. Identify different categories of stakeholder (for example, internal and external (including the more remote) stakeholders).

(Use the Authority, Source and Stakeholder parameters to specify the stakeholders.)

Separating ends and means

It is important to distinguish 'ends' (requirements) from 'means.' 'Means' are the design ideas we choose: the architecture, technology, strategies and other synonyms (They are whatever is needed to achieve the requirements).

It is common to find design ideas included within requirement specifications. I call them 'false requirements'. Only if the design idea is an intentional, conscious design *constraint* should it be in a requirement specification.

All 'false requirements' should be removed from requirement specifications. They should then be investigated; to see if they reveal other hidden additional requirements, which ought to be included (*see the example in Section 2.8. See also Chapter 3, which discusses separation of functions from design ideas*).

Identifying the key requirements

You must try to identify the stakeholder requirements which are either 'vital' (system threatening) or 'profitable' or 'highest risk' for your system. Key requirements have the greatest impact on your most critical stakeholder values and system costs. You do not need (that is, are not economically obliged) to seriously consider implementing any other stakeholder needs than these.

Hints: Look for areas with potentially high development or operational costs. Ask the stakeholders for their opinions on their most crucial requirements.

Note: The concept of identification of the few key requirements (I often use the concept 'Top Ten') does not mean that they will not need to be decomposed into more elementary requirements (see below, Handling Complex Requirements).

> *Remember, for many projects, even delivering a single top objective on time and to financial budget, would be an advance on their current experiences!*

Quantifying success and failure

Requirements need to be understood in terms of success and failure levels. You must ensure you have *quantified* numeric values specified for each of your performance and resource attributes. Knowledge of all the targets ('what we aim for') and constraints ('the limits we need to respect') is vital for both system design and project management. You need to understand exactly what level of achievement is expected and then design towards it. Specifically:

- **Success**: You also need to know when you have met your required levels of requirements. Reaching each single planned level is 'partial' success. Your project is a complete 'success' when all success levels are met, for all performance goals, within all budgets. (*Success levels are targets specified using Goal and Budget parameters.*)
- **Failure**: You need to specify the attribute levels that you have to reach in order to avoid some type of stakeholder failure (such as 'fail to get desired market share'). (*These are constraints stated using Fail parameters. They are not as critical as the Survival constraints*).
- **Survival**: You need to determine and specify the numeric limits which would classify your project as a total failure; so all stakeholders know the minimum survival requirements (*These are constraints expressed using Survival parameters*). These become your highest priority requirements, as they are key to your project's continued existence. Survival is a higher priority than success!
- **Potential**: It is also useful to keep a record of desired, but uncommitted and unbudgeted, requirements. Knowing these, even when you cannot deliver them immediately, is key to being the first one to deliver them when it *does* become possible. (*These are specified using the parameters, Stretch – a deliberate engineering challenge set for the system engineers – and Wish – an expression of the levels which stakeholders 'dream of'.*)

See Chapters 4, 5 and 6, which describe how to quantify performance requirements and resource requirements using the Scale, Goal or Budget (success), Fail (failure), Survival (survival), Stretch (challenge) and Wish (dream) parameters.

Understanding the past and the future – benchmarks and state-of-the-art

You need to understand the context of your requirements. What are the current 'benchmark' performance levels of your existing system and competitors' systems?

Hint: There is always some existing system to usefully benchmark!

Are your plans ambitious enough? Are they state-of-the-art? How do they measure up to your known competitors? How do they fit with current trends in technology? You need to know these factors to understand the level of risk involved and the likely costs. State-of-the-art implies doing something nobody else has yet achieved. This means that costs and success are both uncertain. Don't let that stop you! However, do plan to control this situation rigorously.

See Chapter 4, which describes how to express benchmarks, trends and state-of-the-art levels using the Past, Trend and Record parameters.

Considering the timescales for delivery of requirements

To assess whether your requirements include adequately specified 'time conditions' (dates), you should ask questions, such as: How early are the stakeholders going to receive some benefits from this system? Are the requirements specified for the short term needs only? Are *unrealistically* long investment timescales set?

Most importantly, you should plan the early delivery of some requirements to some stakeholders. There ought to be a steady stream of value delivery throughout the project life.

Hint: Analyzing the requirements of the different stakeholders is one way to identify the opportunities for early deliverables. (See also Chapter 10 on Evolutionary Project Management.)

EXAMPLE One client 'delivered' a mobile telecommunications 'base station' eight months 'early' to its system installers (an internal stakeholder), who were scheduled to install it 'for real', later, in Japan. The installers immediately discovered many serious installation problems, which would have delayed installation. The development project (another internal stakeholder) then had eight months to fix these problems and, not surprisingly, the ultimate system was successfully installed on time (Ericsson, Case Study, 1992, 'On Succeeding', Internal Publication) (Järkvik et al. 1994).

Avoiding the 'ambiguity trap'

Beware requirements that are so 'general' that there is no clear idea of exactly what is required. Everyone can agree to them! For example, 'increase security,' 'make the system more user-friendly' and 'provide a competitive edge.'

The problems due to vague requirements will inevitably arise later, because everyone's interpretation of what the 'general' terms *actually* mean is different. The lack of precise definition means that the differences of opinion are not confronted at an early stage, during specification, and the differences are unspecified. No one has really agreed to the exact requirements and nobody is doing anything about it.

All too often, projects deliberately allow ambiguous specifications to be used, without clarification and agreement. There is the 'illusion of progress being made.' The requirements are 'complete and agreed'; we think?

This problem of 'ambiguous requirements' has to be tackled both by communication and by *action*. Clarifying all the key requirements, as discussed above, helps. However, as a means to get the 'right' requirements, clarification is no substitute for evolutionary delivery (*see Chapter 10*). Frequent and early delivery steps allow stakeholder feedback and correction of bad (vague or irrelevant) requirements and designs. The relevance of the project work to your organization has to be checked: early, measurably and frequently.

Handling complex requirements

Ambiguity ('different interpretations are possible') is one trap. But an entirely different trap exists in losing control of a project because you are operating with *too few* detailed requirements. The degree of detail you will need to specify is dependent on the size and criticality of what you are trying to control, as well as on the degree of risk you are willing to accept.

It is a balancing act. You must keep your attention firmly rooted on the few critical (key) requirements, while ensuring there is adequate background detail to permit you sufficient control. You can do this by specifying a set of complex requirements (the 'Top Ten') and, then splitting each of them into their more detailed 'elementary' components. You then can create any number of useful system views (such as 'Risks', 'Bottlenecks' and 'Progress') with appropriate detail for your project management purposes.

Don't get overwhelmed by the system detail. Capture it. But, always remember to ensure the focus is on your stakeholders' critical requirements.

Allowing requirements to evolve

Real requirements change. There is no way you can stop them! As you run a project or deliver to initial stakeholders, you will get new insights into which requirements are actually useful. Stakeholders, too, will learn from their early experiences using a new system, what they really want. Business requirements will also inevitably change over time, in response to both the internal and external business environments.

For all these reasons, requirements must be *allowed to evolve* during a project, and during the system lifetime. You are not obliged to implement any changes to the system instantly. You can do so at the 'right' time. But, it is essential to keep the *specification* of requirements realistic and up to date. They must reflect current reality. You should not freeze the requirement specification! You can always choose to design, or build or test from a given version of the requirements, temporarily ignoring any updates.

For contractual and other sound reasons, you should always ensure that you document the evolution of requirements (for example, by using automated requirement specification tools that track changed versions).

You need also to build a *web of relationships* between requirements, designs, stakeholders and project plans. This will make it safer to evolve and change, as you will be better able to identify any potentially damaging side effects and to recognize the most competitive change possibilities. (*Planguage offers a wealth of devices for making requirement relationships explicit. For example, by using qualifiers and parameters, such as Authority, Source, Dependency and Impacts.*)

2.2 Practical Example: What is 'Flexibility Improvement'?

Analyzing a requirement

You are told that a change is proposed to 'improve flexibility' within an organization. The stated aim is that it will help you be more competitive by enabling 'faster tailored product releases.' You are not quite sure what this means. You decide to analyze and challenge the statement.

You first give the subject 'improve flexibility' an identity. Call it any name you like. For simplicity (and to show we are addressing the specified concerns), let's call it 'Flexibility.'

This could be written as:

Tag: Flexibility.

However, we usually drop the explicit use of 'Tag'. So it (initially) looks like this:

Flexibility:

To which you could add any relevant information that comes with the idea, or which can be gained by asking key people a few simple questions. For example:

Type: Quality Requirement.
Gist: To improve flexibility of product releases to the market <- Marketing Director.
Authority: Marketing Director request.
Rationale: Supports 'Time to Market'.

Note: The 'Gist' parameter is used to capture a short description of a tagged concept.

Then you can try to write down approximately what you think Flexibility means. Then get others to write down what *they* think it means. Try to get a group to agree to some approximate definition. Write an agreed brief description for the 'improvement ambition level.' The description might come out like this:

Flexibility:
Ambition: Substantial improvement in the ease with which we can change products and markets <- Requirement Owner: Jane.

Note: 'Ambition' is an alternative parameter to use instead of 'Gist' for a quality target (goal). 'Ambition' should express the level of ambition in words.

Now, from this, the *function requirements* can be identified as being to 'Modify Product' and to 'Switch Market.' These are the functions, which we specifically intend to make 'flexible'. (*See also Chapter 3, 'Function Requirements.'*)

We can express these ideas in Planguage as follows:

Type: Function Requirement: {Modify Product, Switch Market}.
Modify Product -> Flexibility.
Switch Market: Supports: Flexibility.

Note: The two formats of the 'Supports' concept are illustrated. The '->' is a keyed icon format.(It is also used as an icon for Impacts.)

Further work can be carried out to establish the precise 'Flexibility' requirements. For example, are completely new products envisaged or is it just the existing products? Are the target markets already established? It is likely that the critical stakeholders already have ideas about where effort is to be directed. Is there a specific current problem or is this a longer term, more global aim?

Next, you can add a statement regarding which of the higher level objectives would probably be impacted, in interesting ways, by improved Flexibility. For example:

Flexibility:
Ambition: Substantial improvement in the ease with which we can change products and markets <- Requirement Owner: Jane.
Supports: Performance: {Time to Market, Market Share, Customer Brand Perception, Product Upgradeability} <- JBG assertion.

Note: '<-' is the 'Source' keyed icon. You should use it to document where any information comes from. '{. . .}' is a convenient way to signal a set of things that belong together in some way.

Also any impacted *cost requirements* should be identified. For example:

Is Supported By: Cost: {Architecture Development Costs, Research Costs}.

Each of these impacted requirements might be considered for expansion into a set of lower level requirements. For example, the quality requirement, Product Upgradeability (*mentioned above*) could be expanded as follows:

Product Upgradeability:
Type: Complex Quality Requirement.
Consists Of:

{Key Upgradeability:
Gist: Improve delivery of <upgrades> meeting <customer> <key requirements>,

Acquisition Upgradeability:
Gist: Increase new product acquisition with aim to supply <customer> <key requirements>,

Customer Installability:
Gist: Improve ability of <customers> to install the <upgrades>, other? }.

Note: words in fuzzy angle brackets (< >) denote words that we feel require additional definition.

This is a simple identification of the various factors, which make up Product Upgradeability. If we agree on them, they can be worked on, to become more specific. The aim is that at some stage, each of these requirements is specified with clear numeric targets that define it more precisely than just using words.

Decomposition of Requirements

It may well be the case that each requirement needs to be expanded into a further set of requirements. These, in turn, may also need expanding resulting in a whole hierarchy of requirements.

At some stage, you identify the requirements that you do not wish to decompose, or you are simply not *able* to decompose, because they are the lowest levels of the hierarchy. Requirements at the lowest level of a hierarchy are termed '*elementary requirements*.' Note: that it is not necessary to identify all the elementary requirements. It is a question of finding the set of requirements, elementary and complex, that best suits your current purposes.

Scalar Requirements

If the requirement concept can be described by 'words implying measurement' (for example 'improve,' 'better,' 'equal to' and 'reduce'), then that requirement is clearly definable in terms of 'degrees of goodness.' Once you have identified such a 'scalar' requirement, the next stage is to improve the definition by quantifying it. You need to find (maybe create) a scale of measure that expresses a unit of measurement for the requirement. Use a 'Scale' parameter to specify your scale of measure.

If you identify several complementary scales of measure, for a single requirement, then you actually have a '*complex requirement*', and you should consider specifying its set of elementary requirements in detail (that is, each elementary requirement has its own Tag and Scale). Note, the set of elementary Scales is the variable 'idea' that describes the complex requirement. The scales of measure within a set don't 'add up'. They don't have to.

Using your chosen scale of measure, you can try to represent the *current and past* levels of performance (the benchmarks) and the desired future states (the performance targets). You do this by defining some specific numeric levels. (*See Chapters 4 and 5 for further explanation.*)

EXAMPLE

Cost to Upgrade Products:
Type: Savings Requirement.
Scale: Total cost, in % of annual profit, needed to develop <new products>.
Past [Last Year]: 4%. "Current level, a benchmark."
Goal [Next Year]: 3% <- Technical Director: JG. "Future desire, a target level."
Defining a scale of measure and using it to specify two points (Past and Goal) to describe the degree of improvement in 'Cost to Upgrade Products.' Note: 1. That although this involves a resource – it is actually setting an organizational performance requirement (an objective) that we need to specifically plan to achieve (by finding relevant strategies). 2. The [...] brackets (qualifiers) are helping to define 'when'.

You don't have to worry about the *exact* truth about requirements if it is not easily available. It never is! But each specification step you take should make things somewhat clearer (even if it is only to help make other *major defects* in the requirements clearer). You should always be totally honest about your uncertainty and about your sources of information. If it seems worthwhile, you can always get more detailed and be more exact at a later stage.

Hint: Discussing your scales of measure with your stakeholders might be helpful.

You may be surprised about our definition of ‘Flexibility,’ but it means exactly what we define it to mean. The tag, ‘Flexibility’ is just an arbitrary reference to the definition (the tag is a ‘symbol’, like all our human words). If you don’t like the tag, change it. How does ‘Product Development’ strike you? You can even have multiple synonym tags for any concept, if that helps you communicate better with different specification readers. Use what works for you!

2.3 Language Core: System Attributes and Requirement Specification Types

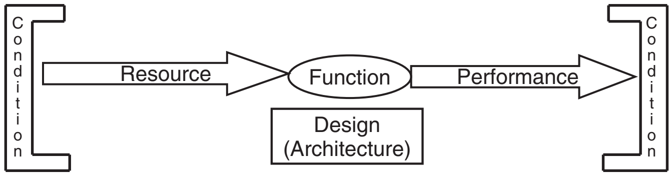


Figure 2.1
The basic system attributes describing a system.

System Attributes

There are several main Planguage specification types that we need in order to describe a system. Let’s now look at the definition of these terms before considering how to specify requirements.

System

A system can be described by its set of function attributes, performance attributes, resource attributes and design attributes. All these attributes are can be qualified by *conditions*, which describe the time, place and event(s) under which the attributes exist.

Attribute

An attribute is a characteristic of a *system*. Any specific system can be described by a set of past, present and desired future attributes.

There are several different types of attribute. These include:

- *Function* attributes defining what a system does (mapping to the processes).

- *Performance* attributes defining how well or how much a system performs (such as usability, availability and response time). How good or how effective it is.
- *Resource* attributes defining *what quantity of resources* a system requires, or what costs are incurred (such as development costs, operational costs and human effort).

  Resources are our necessary and potential fuels, and costs are the experienced or planned expenditures ('budgets') of these limited resources. Resources are a broad category of effort, time, data, materials and money.
- Design attributes, defining the system architecture for a system.

Note: Very often in this book, the term 'attribute' is implied. So 'performance attribute,' 'resource attribute,' 'function attribute' and 'design attribute' become, for short, 'performance,' 'resource,' 'function' and 'design,' respectively.

Function

A function is an *action* of a system or system component. Elementary functions are 'binary' in nature: they are either present, or not, in specifications or in real testable systems.

Each function has a set of *associated* performance and resource attributes, which make it useful and competitive in the real world. However, a 'pure' function is '*what?*' a system does, without regard to either '*how well?*' or '*how much?*' (the resulting performance attributes) or '*what resources?*' (the resource attributes that will be utilized or consumed).

Note: My definition of 'function' is likely to differ from your current definition. I specifically separate the four descriptive system attributes of function, performance, resources and design from each other. My justification is that this separation enables us to obtain better focus within the 'design engineering' process.

Design engineering needs to be able to satisfy many competing performance and resource attributes, simultaneously. Separating the 'multiplicity of concerns' helps identify all the individual concerns; and this in turn, helps ensure they are all considered. This leads to more competitive designs.

If we (mis-)use 'function' in an informal manner, to describe 'designs' and 'features' of a system (which is, unfortunately, common practice), then we fail to see the essential distinctions amongst a 'function requirement,' an 'optional design' or, even, a 'design constraint.' The result is that the design process is corrupted, and weaker designs result since the designer has less understanding of the real design options.

EXAMPLE User Interfacing:
Type: Function.
Gist: All basic user-accessible input and output capabilities within the system.
Note: This does not include the specific system interfaces (the human–computer interface design ideas), which will be developed during the project, based on field trial feedback.
A simple function specification.

Performance

A performance attribute is a 'potential effectiveness' attribute of a system. It is '*how good*' a system is, in objectively measurable terms.

Performance attributes:

- are *valued* by defined stakeholders
- are *always* capable of being specified quantitatively
- are *variable* (along a definable scale of measure)
- can be a *complex* notion, consisting of many elementary performance attributes
- can be *traded off* to some degree, by varying their level, against the resources and/or the other performance attributes. The relative priorities of performance attributes are a question of 'which attributes are more valued' by the defined stakeholders.

Performance levels only partly determine how effective a specific version of a system is, for a specific stakeholder's needs. The practical stakeholder environment determines the 'final' effectiveness that a performance attribute can contribute to. For example, more system speed will not always translate into earlier delivery of specific users' results. And, increasing the average system reliability will not always translate into more reliability, from a specific user's practical point of view.

Another way to express this is that performance in one component of a system does not always translate into the same level of performance in a larger environment. (Compare to the well-known circumstance of the effectiveness of an engine on an icy road or, for that matter, the effectiveness of your mind when put into a noisy environment.)

There are three types of *performance attribute*: quality, resource saving and workload capacity. These are described as follows:

- **Quality**: A quality attribute describes 'how well' a system performs. Examples of qualities are availability, usability, customer satisfaction, staff development, environmental impact and innovation.
- **Resource Saving**: A resource saving is a measure of 'how much' resource is 'saved' compared to some reference or

benchmark system. Resource savings are measures of performance, which describe system costs in relation to alternative costs. They are, you might say, a way of viewing *relative costs* for two systems at once, rather than the absolute costs of one system alone; one system will be the target system and, the other system will either be a past benchmark system or a competitor's system.

EXAMPLE This new car has 10% better fuel consumption than the last model.

EXAMPLE The cost per transaction for System X [New Version] might be 100 dollars, but the savings for System X [New Version] might be expressed as '50% less cost' compared to System X [Last Version], which cost 200 dollars per transaction.

Other examples of resource savings include:

- *operational* savings of any resource (such as effort, money, time, materials and space)
- *capital investment* savings (say, for activities such as for launch, training, installation and acquisition).

- **Workload Capacity or Capacity**: A workload capacity attribute describes '*how much*' a system can do. Workload capacity describes the *potential* workload a system can tolerate.

Workload capacity attributes include:

- Throughput capacity: how much work can be done
- Storage capacity: how much information can be contained
- Responsiveness: how fast the system responds.

Resource

A resource is a system 'input fuel' attribute.

Resource is used as follows:

- to 'start up' or get a system going – expending a 'capital cost' – investment
- to keep a system functioning (using or expending a resource is an 'operational cost')
- to bring about change in a system (expending a development or maintenance cost).

'Cost' is the degree of a resource used (a cost benchmark) or planned to be used (a cost *budget* or resource *budget*). For example: time, work-hours, talented people, investment capital, staff costs, development costs and operational costs.

Resource attributes:

- are capable of being specified *quantitatively* (for example, 'resource use limits' and 'cost plans')
- are *variable* (along a definable scale of measure)
- can be a *complex* notion, consisting of many elementary resource concepts
- can have *complex resource targets* specified (there can be specific resource allocation, using 'qualifiers', regarding when, where and under which events it can be used)
- can be *traded off*, to some degree, against the performance attributes and/or the other resource attributes.

Note: Many characteristics of a resource attribute are identical to those of a performance attribute. The difference is that one is a 'means' (resource) and the other is an 'end' (performance).

Design

The design of a system is also an observable system attribute. You can look at any system and ask, "What is its design?" This knowledge is useful for the following reasons:

- it explains how to *reproduce* the system
- it can explain the *current performance levels and cost levels*
- it can give you insights as to the *ease of making* specific *design changes* or the need to *upgrade specific components.*

The design of a system can be specified at any number of levels: from high-level strategies and architecture to low-level, detailed system components. The precise terminology used is a matter of culture and taste: a design attribute is anything that impacts the functionality, performance and/or costs of a system.

In Planguage, a system *design* is modified by implementing a series of Evo steps. Each Evo step can have one or more *design ideas.* A 'design idea' is the primary output of a design process. It is the generic name for any proposed design strategy, or system component, that we need to identify, to specify, to analyze and perhaps to implement in order to address the problem of reaching our stakeholder requirements. More simply: design ideas are our 'tools to reach our ends.' It is any idea or strategy, which possibly contributes to the 'design solution.'

Requirement Types

Above, we looked at a system (or project) from a descriptive point of view. This is also the benchmark view of a system, a view that we will integrate with the specification of requirements. Below, we shall look at the same concepts in terms of how to specify what we want in the future.

Vision

At the highest level, there should be a vision statement for a system. A vision or vision statement is a specific, long-range, overall category of requirement. That means it can concern itself with future mission and/or targets and/or constraints. It is a leadership statement for focus and motivation. Visions are often defined in broad summary terms. For example, 'become world class.' But there is no reason to be so vague. Great practical visions[1] are extremely concrete:

"*I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to the earth.*"

John F. Kennedy.
Delivered before a joint session of Congress, May 25, 1961.[2]

"*I believe that we must improve the numeric level of all critical product and service qualities by an order of magnitude by the end of the decade in order to remain competitive.*"[3]

John Young,
CEO Hewlett Packard Company, April 1986.
Known at the '10X' policy.

"*We shall go on to the end, we shall fight in France, we shall fight on the seas and oceans, we shall fight with growing confidence and growing strength in the air, we shall defend our island, whatever the cost may be, we shall fight on the beaches, we shall fight on the landing grounds, we shall fight in the fields and in the streets, we shall fight in the hills; we shall never surrender.*"

Churchill, June 4, 1940.[4]

A vision will ultimately need to be decomposed into specific requirements such as measurable objectives with quantified goals. Using qualifiers, these requirements can, as necessary, be tied by specification to specific times, locations, components and events of the system.

[1] See also the Martin Luther King Jr. vision in the Glossary under Vision.
[2] FROM http://www.jfklibrary.org/, Special Message to the Congress on Urgent National Needs, President John F. Kennedy, delivered in person before a joint session of Congress, May 25, 1961.
[3] This is the best rendition available in consultation with HP – Tom Gilb.
[4] *The Oxford Dictionary of Quotations*, Third Edition with Corrections 1980. Oxford University Press.

EXAMPLE Vision 1:
Vision [By Next Year, Software Products]: 30,000 hours mean time between failure <- CEO in last Annual Report.
"30,000 hours mean time between failure for software products by next year."

EXAMPLE Vision 2:
Vision [Within Next Three Years, Key Products]: Order of magnitude reliability improvement <- Technical Director.
"Order of magnitude reliability improvement in our key products within three years."

Once a vision is in place, specific strategies/design ideas can then be evaluated against it, as potential solutions.

(*See Glossary or Chapter 3, 'Functions,' for discussion of 'mission.'*)

Basic Requirement Types

Once you have a vision statement providing the overall direction for a system, you can start capturing the specific requirements for change.

There are the following basic requirement types:

(*These basic requirement types have been already outlined in the introduction to this chapter; here they are discussed in more detail.*)

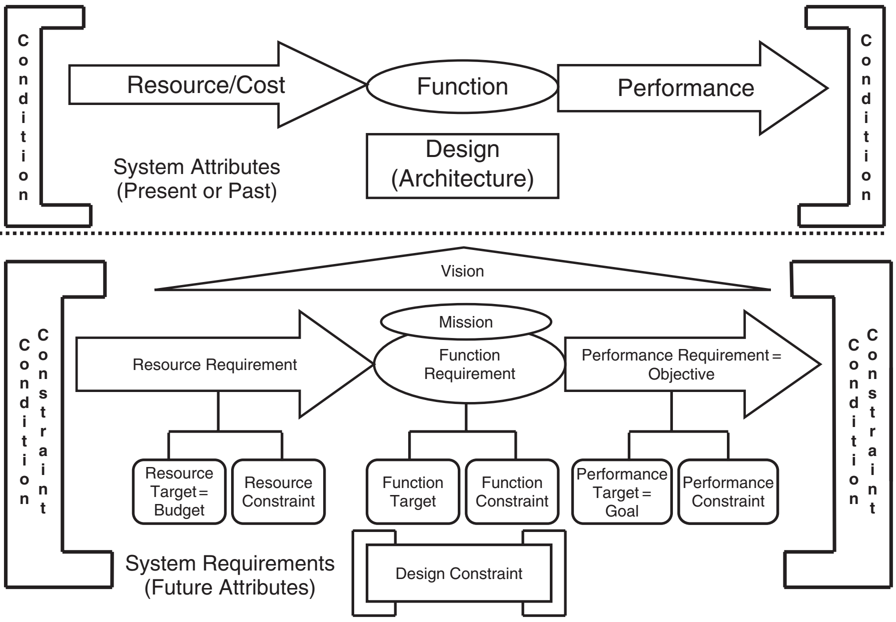


Figure 2.2
Mapping of system attributes to requirements.

1. **Function Requirement: what a system has to 'do.'**
   Function requirements are the functions that are fundamental to the system, the marketplace or the contract we have undertaken. In competitive product areas, the functionality defines 'the market we are in,' such as 'Producing mobile phones.' All our competitors probably have identical functionality.

   Note, 'functionality' does not mean design features and quality; it means pure basic function. The competitive product differentiators are the performance levels and costs, not function. A function requirement is a function that is either declared by the stakeholders to be required, or is formally recognized by all stakeholders as a *fundamental* function of a system.

   Function requirements can provide a framework, rather than simply stating the precise functions required. A function requirement could specify some set of functions (for example, 'All Competitor X functions'). It could also specify functionality that is not required, (for example, 'No Games').

   (*See Chapter 3 describing Functions and Function Requirements.*)

2. **Performance Requirement**: 'how good' a system has to be.
   A performance requirement is also known as an **objective**. All performance requirements are 'scalar' (meaning numerically 'variable') in nature and must be specified quantitatively. That is, there should be a defined scale of measure (*Scale*) and a specification of the future required numeric levels for success, failure-avoidance and survival (*Goal, Fail and Survival parameters, respectively*) with relevant conditions (*the [time, place, event] qualifiers*).

   The minimum specification for a 'performance requirement' is that there must be one target (a *Goal, Stretch or Wish level*) or one future **constraint** (a *Fail or Survival level*). Of course, any number of useful targets and constraints can be specified.

   Finally, benchmark information is needed to complete any requirement specification. Without such a 'baseline,' there is no way to understand the relative ('improved') change required. So a complete performance requirement specification will include at least one benchmark (a *Past, Record or Trend level*).

A performance *requirement* specification can consist of:

- a set of *targets* (*Goal, Stretch and Wish levels*) and
- a set of *constraints* (*Fail and Survival levels*).

and is supported by:

- a set of *benchmarks* (*Past, Record and Trend levels*).

Note: As a performance requirement is scalar, all these are **scalar** *parameters.*

There are three kinds of performance requirement:

- quality requirement
- resource saving requirement
- workload capacity requirement.

2.1 Quality Requirement

A quality requirement expresses '*how well*' a system will perform.

EXAMPLE

Adaptability:
Type: Quality Requirement.
Scale: Time in hours needed to re-configure the defined [Base Configuration] to any other defined [Target Configuration] using defined [Methods] and defined [Reconfiguration Staff].
Expert Reconfiguration: Defined As:
{Base Configuration = Novice Setup,
Target Configuration = Expert Setup,
Methods = Selection of Library Reconfiguration Process,
Reconfiguration Staff = Qualified Expert}.
========================== Benchmarks ==========================
Past [Expert Reconfiguration, Version 0.3, Asian Market]: < 1 hour.
======================= Performance Targets =======================
Authority [Goals]: Federal Drug Administration.
Goal [Expert Reconfiguration, Deadline = Version 1.0]: < 0.5 hours.
Goal [Expert Reconfiguration, Deadline = Version 2.0]: < 0.1 hours.
========================== Constraints ==========================
Fail [All USA Products]: < 0.7 hours.
Fail [Expert Reconfiguration, Deadline = Version 2.0]: < 0.5 hours.
Survival [Expert Reconfiguration, European Market]: < 1 Working Day.
This quality requirement is a measure of how well a system is designed to adapt to reconfiguration needs in the future.

Note:

- *This is a quality requirement even though it has a Scale that involves measurement of a resource. The reason that this is not a resource saving is that a **specific level** of resource saving is not being requested. The resource measurement is simply a convenient way of capturing the 'adaptability' of the system.*
- *This is also not a budget (a resource or cost requirement) as the level is not set up primarily to monitor the expenditure of resource.*

These are important distinctions, because as a consequence of them, you will be forced to react in different ways to the problems arising in meeting these requirements.

2.2 **Resource Saving Requirement**

A resource saving requirement defines some required level(s) of saving of a resource compared to the benchmark system(s). *How much resource do we have to save?*

EXAMPLE

Customer Installation Cost:
Ambition: Reduce the costs to our customers of installing our products on customer sites.
Type: Resource Saving Requirement.
Scale: Average Total Installation Cost for each Installation of defined [Product] for all <involved customer departments> within defined [Customer].
Total Installation Cost: Defined As:
{Cost of Education of <customer people>, Cost of Involvement during Planning of <customer people>, Cost of Shipment of Product, Cost of Involvement during Installation of <customer people>}.
PP: Past [Last Year, Customer XYZ, Product ABC]: Average <worldwide> Total Installation Cost for each Installation of Product ABC for Customer XYZ expressed in $.
Fail [For each Installation, USA, Release 1]: PP.
Goal [For each Installation, USA, Release 1]: 80% of PP.

2.3 **Workload Capacity Requirement**

A workload capacity requirement defines one specific capacity of a system for doing work. It specifies an aspect of '*how much*' work a system or product will be expected to perform in operation.

Capacity requirements cover such things as transaction speeds, data storage, maximum transaction volumes and maximum concurrent users.

EXAMPLE

Responsiveness:
Ambition: Fast immediate response to any type of user asking for information.
Type: Workload Capacity Requirement.
Scale: Time in seconds from when a defined [User] knows what they want to ask until the correct necessary information is available to them to carry out a defined [Task].
Past [User = Free Set, Task = Inquiry]: Over one minute. Note: Considered unacceptably slow.
Goal [User = Responsible Administrator, Task = Any Administration Task]: under 5 seconds? <-Guess TG.
Goal [User = Phone User, Task = Call Setup]: Less than <2 seconds?> <- RB.
Note: Depends on type of call you want to set up.
Example from a client specification (edited).
(*See Chapter 4 describing Performance and also Chapter 5 on Scales of Measure.*)

3. **Resource Requirement**: how much a system can cost
   A resource requirement (budget) is a cost (expenditure) requirement. A budget is a plan for the use of a finite resource. A budget is a statement of stakeholder-imposed:

   - resource targets (Budget, Stretch and Wish levels)
   - resource constraints (Fail and Survival levels).

   How much of a limited resource do we plan to use?

   Like performance requirements, all resource requirements are 'scalar' or variable in nature and must be specified quantitatively.

   We are interested in specifying resource requirements for two closely related purposes. One is so that the design process can 'design to cost.' The other purpose is to help us influence the performance to cost ratio. Ultimately, it is the benefit to cost ratio of any product, organization or system, which defines its competitiveness in the marketplace. Of course, we must control both performance and its costs simultaneously. (*See Chapter 6 for further discussion on Resources.*)

4. **Design Constraint**
   A design constraint is an explicit and direct restriction regarding the choice of a design idea (This includes any architecture or strategy).

EXAMPLE

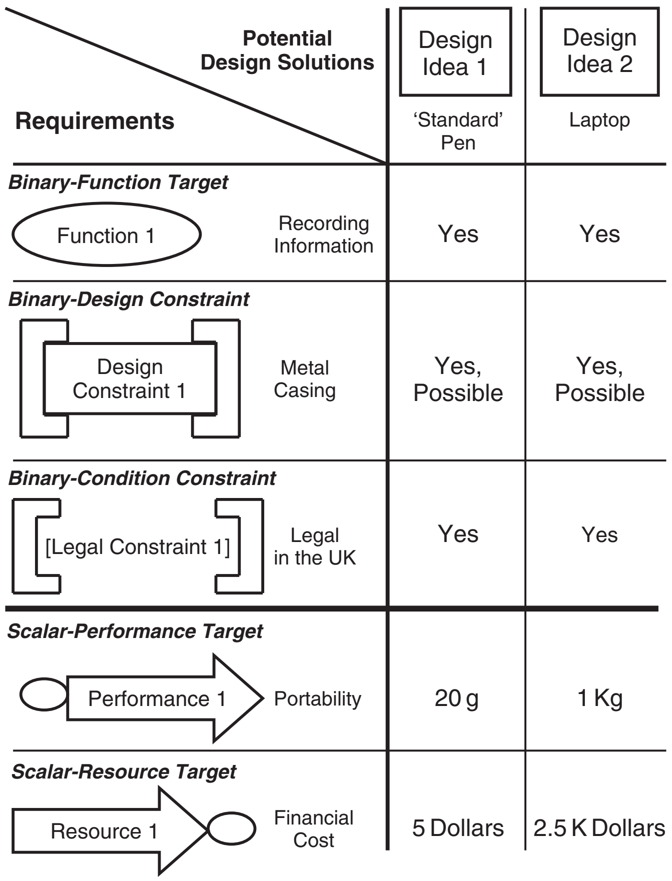
Euro Safety Design [European Models]:
Type: Design Constraint.
Description:
Use designs {X, Y, Z},
Do not use designs {M, N, P}.
Authority: European Safety Law.
Responsible Manager: Corporate Safety Director.
Implementer: Product Line Architect.

5. **Condition Constraint**: *what restrictions are imposed?*
   Condition constraints are restrictions on the system lifecycle – that is, on the system design, operation or disposal – *other* than those constraints expressed as attribute constraints (that is, other than those expressed as function constraints, performance constraints, resource constraints and design constraints). A condition constraint may be expressed as a qualifying [time, place, event] condition or by using a Constraint parameter.

   All condition constraints are binary (non-scalar). A condition is either fulfilled or it is not. A condition is either true or false.

There is a potentially very long list of classifications for the condition constraints. For example: Legal Constraint, Political Constraint and Cultural Constraint. Classification is not essential, just useful. They are what they say they are in the specification.

Condition constraints *can* impact the design choices of a system. That is, an architect or a systems engineer is free to choose any design that does not violate the constraint.



Potential Design Solutions / Requirements		Design Idea 1 'Standard' Pen	Design Idea 2 Laptop
Binary-Function Target Function 1	Recording Information	Yes	Yes
Binary-Design Constraint Design Constraint 1	Metal Casing	Yes, Possible	Yes, Possible
Binary-Condition Constraint [Legal Constraint 1]	Legal in the UK	Yes	Yes
Scalar-Performance Target Performance 1	Portability	20 g	1 Kg
Scalar-Resource Target Resource 1	Financial Cost	5 Dollars	2.5 K Dollars

Note:

1. The above table includes the binary requirements, which are not normally shown. (Usually, all the design ideas are informally screened against the binary requirements before drawing up an IE table. An IE table typically only shows the scalar requirements.)
2. The table is without the scalar baseline information that states the quantitative requirements and benchmarks, which permit percentage comparisons and improved design idea evaluation.
   See Chapter 9, Impact Estimation, for further explanation of IE tables.

Figure 2.3
This is a modified form of an Impact Estimation (IE) table showing an arbitrary set of requirements and two potential design ideas.

Table 2.1 Planguage Architecture. See the Glossary for further information; this table only contains the main concepts.

Planguage Architecture Parameter Class	*Parameter Name, Type or Content*	*Use*	*Notes*
Specification Control	Tag Version Specification Owner Status Quality Level	Administration and Authorization of specifications.	Version can be used at the level of the individual specification object, not just at document level. Documents are 'reports' of views from specification databases of specification objects.
Stakeholder Role (Agent)	– Consumer/Customer/ Product User – Client/Customer/ Product Business – Customer Manager – System Owner – System Designer – Specification Author – Project Manager – System Tester – System Maintenance – Authority – Sponsor – Funder – Champion – Other	Specifies role played by individuals or organizational groups. Stakeholders can be internal or external to a specific system.	Provides information about the nature of responsibility and the relationship to a specification.
Scope	Scope Properties: – Global/Local – Generic/Specific – Internal/External (Inside or outside a specified scope)	Defines applicable specification/system space. See 'Condition'	Answers the question of 'How influential is a specification/system?' Defined using [time, space, event] conditions.
Condition	When – Time Where – Place Where – Place by Stakeholder Role or Organizational Group Where – System Component If – Event Defines a system attribute, a system requirement or a potential system design. Requirement Design Idea	Defines scope (space dimensions) and, indirectly, priorities. All these objects can be complex or elementary.	Declared using Qualifiers [. . .] or a Condition parameter. A complex object can be decomposed into a set of elementary objects.
System Attribute (Attribute)	Function Performance: – Quality – Resource Saving – Workload Capacity Resource/Cost Design/Architecture	'Function' includes 'Mission' at the highest function level.	Often simply referred to as 'Attributes'

Continued

Table 2.1 Continued

Planguage Architecture Parameter Class	*Parameter Name, Type or Content*	*Use*	*Notes*
Requirement	0. Vision 1. Function Requirement 2. Performance Requirement (or Objective) – Quality Requirement – Resource Saving Requirement – Workload Capacity Requirement 3. Resource Requirement 4. Design Constraint 5. Condition Constraint	To specify and agree stakeholder needs	The primary competitive ideas are the performance requirements.
Attribute Class	Benchmark/Baseline Target Constraint	Declares/clarifies intended use of the specification.	
Benchmark/ Baseline	Past Record Trend	Systems Analysis. Compare to requirements: targets and constraints.	Analysis data is integrated with other
Target	Goal (for Performance) Budget (for Resource) Stretch Wish	Defines a numeric value, which is valued by stakeholders	Must be considered together with the [qualifier] information to be fully interpreted.
Constraint	For a Scalar Constraint: – Fail – Survival For a Binary Constraint: – Constraint (Usually with an adjective, such as 'Function,' 'Design' or 'Legal')	Defines a limit for a numeric value or certain specific criteria, which has to be respected to avoid failure or worse.	Stakeholders impose constraints. Given the same set of qualifiers, constraints are of higher priority than targets.
Standards	Policy Rule Process – Entry Condition – Procedure – Exit Condition Interface Template Form Other	Defines Work Process Standards.	Specification rules define the concept of 'defects' in a specification. This enables quality control, process control and process improvement. Either specification standards or system standards. For requirements and much else.
Specification	Requirement Specification Design Specification Architecture Specification IE table Evo Step Specification Evo Plan Systems Architecture Standards Specification		

2.4 Rules: Requirement Specification

Tag: Rules.RS.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Base: The rules for generic specification, Rules.GS apply. For the different types of requirement use also the relevant rules (that is, Rules.FR, Rules.SR and Rules.SD).

R1: **Stakeholders**: There must be a list of the defined stakeholders and it must span the entire product lifecycle and system space.

For any specific specification, the specific stakeholders can be stated or defined explicitly. For example, Stakeholders: {A, B, C}.

R2: **Scope**: The scope or 'system space' of the requirements must be defined. All specified qualifiers for requirements must be relevant to the system space.

Note: Scope states the 'overall system boundaries'. The scope for specific requirements is generally specified using [qualifiers]. See Section 2.7 for discussion of qualifiers. Use a Scope parameter if you want an explicit definition.

R3: **Qualifier Conditions**: Using qualifiers, requirement specifications must adequately cover the time period (When: long term and short term) and the physical scope (Where) for the system and, must state any known dependency on conditional states or events (If).

R4: **Rationale**: The rationale or justification for a requirement and for specific aspects of it should be given. *Use the Rationale parameter.*

R5: **Dependencies**: Any conditions or circumstances, which a requirement depends on for relevance or authority, must be specified. (*Use the 'Dependency' parameter, or any other relevant means.*)

R6: **Internal Links**: All specified requirements can be grouped into relevant hierarchical levels of requirements. Linkage to related requirements should be explicit and complete.

For example, use Planguage specifications such as:

- *Hierarchical tags (for example, 'System.Subsystem.Component').*
- *'Consists Of' or 'Includes' to link to lower hierarchical levels.*
- *'Is Part Of' to link to higher hierarchical levels.*
- *'Supports' and 'Is Supported By' to explicitly specify any intended direct links.*
- *'Impacts' and 'Is Impacted By' to explicitly specify impacts including any side effects (Impact Estimation table linkage).*

R7: **External Links**: Requirements which are related to any level of product line requirements, corporate standards or policies, or anything outside of the specific system documentation, must always explicitly indicate that relationship by a suitable specification (By use of parameters, such as *Supports and/or Impacts*). The intended readership should not have to know or guess such relationships (for example, shared interfaces, shared objectives and use of generic templates).

R8: **Testable**: Each requirement must be specified so that it is possible to define an unambiguous test, to prove that it is actually implemented.

A specific test may be specified or outlined immediately in the Meter or Test statement. However, any specific tests will usually be designed in detail later. The key idea is that all requirements must be clear enough to be testable by some means.

R9: **Design Separation**: Only design ideas that are intentionally 'constraints' (*Type: Design Constraint*) are specified in the requirements. Any other design ideas are specified separately (*Type: Design Idea*). *All the design ideas specified as requirements should be explicitly identified as 'design constraints' (that is, 'design ideas' which are 'constraints').*

2.5 Process Description: Requirement Specification

Requirement specification is carried out throughout a project's lifecycle. It occurs when specifying the initial overall top-level requirements and, subsequently, during each evolutionary result cycle. (Within each evolutionary result cycle, the top-level requirements are reviewed, and updated if necessary, and the subset of requirements relevant to the specific step is specified in detail.)

A generalized requirement specification process is given in this section. Specifically, it does not include any detailed review or updating considerations.

Process: Requirement Specification

Tag: Process.RS.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Entry Conditions

E1: The Generic Entry Conditions apply. The Specification Quality Control (SQC) entry condition applies to any source information, such as contracts and marketing plans.

E2: Key stakeholders should be available for questions and reviews to resolve any uncertainty about sources and exact specification.

Procedure

P1: Define the system scope and the overall scope of the requirements.

P2: Identify relevant (critical and profitable) stakeholders.

P3: Determine the requirements of each type of stakeholder. Ensure all specification statements are source-referenced.

P4: Categorize requirements by type (the major requirement types are function requirement, performance requirement, resource requirement, design constraint and condition constraint).

P5: Specify *Function Requirements* (Process.FR. See Chapter 3).

P6: Specify *Performance Requirements* (Process.PR. See Chapter 4) including identifying or creating a Scale of Measure (Process.SD. See Chapter 5).

P7: Specify *Resource Requirements* (Process RR. See Chapter 6).

P8: Identify and question any design constraints and condition constraints. (Are they real or was something else intended?) Ensure the necessary *design and condition constraints* are specified.

P9: Specify all known significant relationships of the requirements to any other relevant requirement specifications (external or internal to the system). *You need to identify where there may be overlap or conflict or double accounting over benefits. There may even be synergy or a chance to 'subcontract' parts of the system development.*

Use Planguage terms such as {Source, Dependency, Assumptions, Authority, Impacts, Risks, Is Impacted By}.

P10: Get stakeholders to approve the written requirement specifications that specifically affect them.

P11: Carry out Specification Quality Control (SQC) on the requirement specification.[5] Obtain management review approval.

[5] For the majority of the procedures in this book, the exit and entry conditions serve to remind you about the need for quality control: explicit reference to quality control within the main procedure is usually omitted.

Use sampling to obtain information about the likely number of remaining major defects/page. An appropriate default, general exit condition is a maximum of one remaining major defect/page (300 non-commentary words).

Note: This is an appropriate point in this procedure to carry out quality control. However, don't let this prevent you from carrying out quality control at other times. For example, it is far better you find out that there is a problem after writing three pages than after 30 pages.

Exit Conditions

X1: The Generic Exit Conditions apply. The requirement specification must have exited SQC.

X2: There is management review approval of the requirement specification.

Note: This exit does not mean that the requirements can or should be 'frozen' and final. They are merely ready for continuous refinement, detailing, correction and supplements, which will result primarily from feedback from early and frequent evolutionary delivery steps.

2.6 Principles: Requirement Specification

1. **The Principle of 'Results Beat All'**
   The top strategy is 'getting the stakeholder results'.

   Meeting requirements is more fundamental than any other process or principle.

2. **The Principle of 'Goodies Control beats Bean Counting'**
   Focus on getting the Goodies. Their costs will be forgiven.

   The main point of any project, or change effort, is to improve stakeholder benefits. The benefits must be at least as well-controlled as the resources needed to get them. Otherwise the benefits will lose out, at the hands of the always limited, clearly budgeted resources.

3. **The Principle of 'Reasonable Balance'**
   Reach for dreams, but don't let one of them destroy all the others.

   You cannot require an arbitrary set of requirements. There must be balance between performance requirement levels, resources available and available design technology.

4. **The Principle of 'Unknowable Complexity'**
   You must feed a lion to find out how hungry it is.

You cannot have correct knowledge of all the interesting requirements' levels for a large and complex system in advance. You cannot know which requirements are needed, and which are realistic, until you have some practical experience with a real system with real people using it.

5. **The Principle of 'Specification Entropy'**
   Even gourmet decays.

   Any requirement or design specification, once made, will become gradually less valid, as the world, for which they were intended, will change over time.

6. **The Principle of 'Critical Values'**
   If you don't find the critical requirements, they will find you!

   You must identify all potentially requirements for all stakeholders or you risk losing profitability, or even system failure.

7. **The Principle of 'How Good' and 'How Much' before 'How'**
   All performance requirements and resource requirements must be stated before any design idea can be fully and *properly* evaluated.

8. **The Principle of 'Gap Priorities'**
   The least fulfilled requirement attributes become our current priorities.

   By calculating the 'gap' between current real levels of performance delivered and the required levels, we can assume that the biggest unfilled 'gap' in meeting our targets is our current greatest priority. For example, you cannot know now if you will be hungrier, thirstier or more tired a week from now. But wait a week and you will know which need has priority.

9. **The Principle of 'Stop the World, I Want to get Off'**
   There is no final set of real-world requirements; freezing the specifications will make your real problems worse than any problems caused by updating them.

10. **The Principle of 'Eternal Projects'**
    Survival is a lifetime project.,

    The process of delivery of results has no end, if you are in competition for survival.[6]

2.7 Additional Ideas

Using Qualifiers to Specify Conditions

Planguage is able to capture a wide variety of situations. This capability allows us to target specific parts of a system; for example,

[6] Based on the wisdom of W. Edwards Deming.

aiming to deliver to our most critical stakeholders and customers early, without waiting for the entire systems effort to complete. The major tool we use to give this flexibility and power is the 'qualifier' statement.

Qualifier Definition

A qualifier specifies any useful set of conditions that must be fulfilled, in order for the specification to become effective (valid as a requirement, a design or other specification). The qualifiers usually specify when, where and under what special conditions a specification is valid. There are three main classes of conditions [time, place, event], in other words, [when, where, if]. They are specified as follows:

- time' or 'when' states a time concept.
  - This can be a date. The date can be past, present or future.
  - It can also be any relative notion of time, such as [After Release 1].
  - It can be any multiple notions of time. For example, [After April 1, Except Sunday].
- place' or 'where' states a notion of 'placement'.
  - 'Where' stated as a 'physical location' has a wide range of interpretation; it can be any component *part* of a system and/or any *physical location* where the system operates or has operated or will operate.
  - For example: [Market = European Union],
    [Use Area = At School],
    [System Module = {Module A, Module B, Module F}].
  - The 'where' location can even be stated indirectly by reference to any aspect of the system that implies certain areas. For example, 'where' can be captured by naming the stakeholders involved (by user roles, or by their relationship to specific locations), or tasks.
  - For example, [Stakeholder = {First Time User, Pupil}],
    [Users = Account Managers],
    [Users = Head Office Staff],
    [Task = Address Entry].
- 'event' or 'if' states any *special circumstances* that have to be in a 'true' state for the specification to apply (For example, [If Contract23 = Signed]).

  (Aside: This final category of 'event'/'if' is really a somewhat simplified concept. The main aspect to consider is capturing any 'special circumstances/conditions.' If you think about it, all conditions, including time and place are actually 'if' conditions.)

Qualifiers are defined within a Scale definition or within an individual Planguage statement on a 'need to know' basis.

Qualifiers defined within a Scale definition are known as 'Scale Qualifiers.' When using a Scale, all the scale qualifiers have to each be assigned a 'Scale Variable.' A scale variable can be assigned by default value, by explicit declaration or by implied inheritance.

EXAMPLE

Training Time:
Scale: Average time in minutes for defined [User: default = Student] to complete defined [Task].
Goal [User = Year 1 Student, Task = Learn to Use Library Catalogue, School = G&L]: 10.
Referring to potential qualifiers in a Scale definition using Scale Variables. User and Task are defined within the Scale and are Scale Qualifiers. 'Student' is a default Scale Variable for User. School is added in the Goal (performance) statement as an additional qualifier.

Qualifiers are usually stated within square brackets. However, there is also a Qualifier *parameter*.

EXAMPLE

Goal [Case Home]: 99.5%, [Case Euro]: 99.6%.
Source: Product Planning.
Project Defaults: Qualifier [Years End, Consumer Goods, If Fierce Competition on Price].
Case Home: Qualifier [Home Market, Project Defaults].
Case Euro: Qualifier [Euro Market, Project Defaults].
A qualifier statement can be defined independently, for example in order to reuse it, or to have a short summary reference to it elsewhere.

There is no sequence requirement for the conditions. There can be multiple instances of any one class of condition. For example: [Country = {USA, UK, NO}].

The qualifier content should either be self-evident for purpose (For example: [End of this Year, USA, If No War] or make use of additional explicit qualifier parameters as follows: [Qualifier Name = Qualifier 'Value'].

EXAMPLE

Goal [Deadline = End of Next Year, Country = UK, State = If No War]: 55%.

Qualifiers can be present in any requirement, design idea, or Evo step specification. Most Planguage parameters can use qualifiers: certainly all benchmarks, targets and constraints would be *expected* to have qualifiers present.

In fact, without adequate qualifiers, a specification is too general. For example, for a requirement to really exist, time and place conditions must be set.

Qualifiers can apply 'by default' from other system specifications. This is called 'inheritance'. Inheritance occurs from more global specifications and/or from higher hierarchical specification levels. In such situations, there exist no 'more local' qualifiers that override the inherited qualifiers.

Qualifiers and System Space/Scope

Scope is the overall 'space' for a system. The scope for specific requirements is generally specified using [qualifiers]. Alternatively, you can use the Scope parameter if you want to state a set of scope boundaries as a separate reusable statement.

Constraints may help establish the limits of the system scope (boundaries). Condition constraints can be used to specify any specific conditions that are limits to the scope of a system.

The Difference between Qualifier Conditions and Condition Constraints

Qualifier conditions are not usually constraints. Any specification (such as requirement, design, implementation planning or test planning) can contain qualifier conditions of any kind. Qualifier conditions must all be 'true' for the related specification to be made effective. The effective specification may or may *not* itself be a constraint specification. (A constraint sets a limit because some kind of 'pain' will be experienced if the constraint is not met/conformed to).

EXAMPLE L [X, Y]: Type: Condition Constraint: The system must be legal in area E.
G [M, N]: Type: Function Requirement: Children's Games.
L is a condition constraint, which is activated only when qualifier condition X and Y are both true.
G is NOT a condition constraint. It is a function requirement that is a valid requirement when both conditions M and N are true.

Qualifiers and Evo Steps

One of the many uses of qualifiers is in helping us to 'divide up' both requirements and design ideas into 'chunks' for implementation purposes. All qualifiers specified in requirements help identify potential 'natural boundaries' within the system that might enable sub-setting of the system to support selection and delivery of Evo steps.

Deadlines provide a set of time sequences, and 'place' qualifiers give a set of locations that can be exploited in planning the evolution of the system. Even an *event* condition can give us the possibility of further differentiation for selection of possible Evo steps.

Additional Ideas Concerning Constraints

Constraints are not the main reason our project exists and they are certainly not what we are investing in. However, constraints are essential requirements as they provide the information about the design limitations, which we must adhere to: the absolute limits for performance and resource levels and, the absolute restrictions on what we can and can't do.

Constraints are set as a result of many factors: corporate policy, national laws, competitive forces and limited project resources, to name a few of the many areas that supply us with constraints. The penalty for us if we do not identify, specify and respect these constraints is some degree of partial, to total, failure to deliver the stakeholder requirements. Constraints are not 'fun,' but try to think of them as presenting interesting engineering challenges.

Adherence to Constraints

When designing a system, the list of constraints needs to be treated as a checklist against which every single potential design idea has to be checked for possible violation. Remember also to check any sets of design ideas and, the potential total design (if it is outlined) against the constraints. It could be that collectively a set of design ideas violates some constraint(s). For example, by exceeding a resource constraint. Any potential design idea that violates any constraint might be rejected for this reason. But, not for sure! It depends on the relative priority of the requirements, which the design idea is trying to satisfy, as well as the options for alternative design ideas. In some cases, the constraint itself may have to 'back down'. It would be good practice to specify what has happened in the design specification.

EXAMPLE Note: This design conflicts with the following constraints {CA, CB}, but we have decided to make an exception, as no other better alternative has been found <- TG. Authority: Chief Architect.

One point to bear in mind is that constraints always result from the choices of stakeholders. What might be a 'given' constraint to you is likely to be the free choice of another stakeholder. If you decide there is an issue with a constraint or that a conflict exists, then the first thing to consider is the authority that 'set' the constraint. You can then determine how to treat the issue to achieve resolution.

Remember, constraints have cost implications as well: the addition, alteration or removal of a constraint can have significant impact on the implementation or operational system costs.

EXAMPLE C1: European Community Suppliers of <system components> must be used, where possible.
Type: Political Constraint.

EXAMPLE C2: The system must be legal in the country of operation.
Type: Legal Constraint.

EXAMPLE C3: It cannot cost more than 'Y' to develop <the system>.
Type: Resource Constraint [Resource = Financial, Lifecycle Stage = Development].

EXAMPLE C4: It cannot cost more than 'X' to produce, distribute or support <the system>.
Type: Resource Constraint [Resource = Financial, Lifecycle Stage = Post Development].

Constraint Viewpoints

Constraints can be classified from several viewpoints (*see Figure 2.4*). If you consider the system lifecycle viewpoint, two specific categories

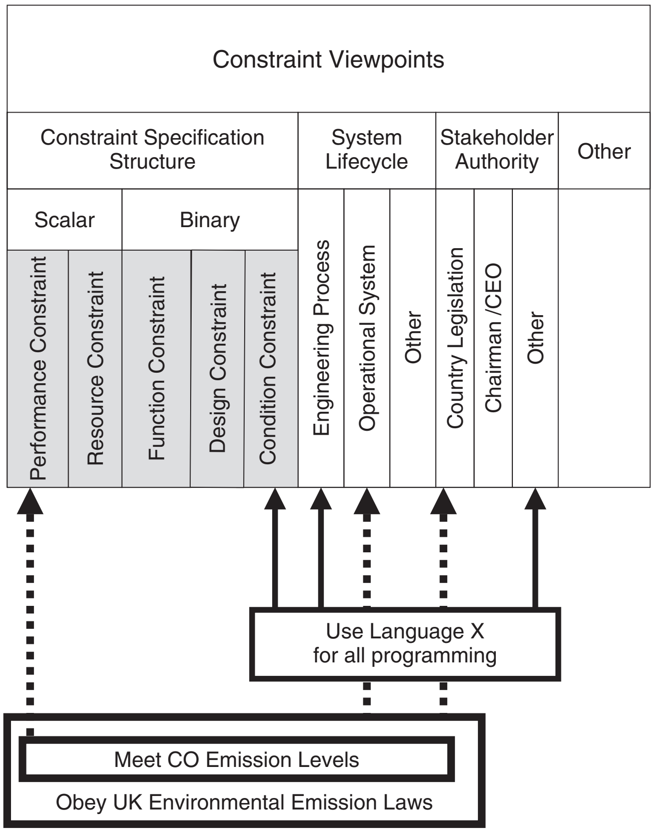


Figure 2.4
Constraint viewpoints: constraints are either scalar or binary. They can be categorized from several viewpoints.

of interest could be ‘System Operational Constraints’ and ‘Engineering Process Constraints.’ System operational constraints apply across the entire operational system, while engineering process constraints only restrict the engineering process itself (as opposed to the system being engineered). A constraint such as ‘Meet carbon monoxide emission levels’ is simultaneously a system operational constraint, a performance constraint and a legal constraint.

2.8 Further Example/Case Study: A Proposal to the Board for $60 Million

Here is the original plan (edited to conceal identities) presented to the Board of Directors of an engineering organization, requesting $60 million for CAD/CAM equipment. It was written by the Engineering Director for Quality and Productivity. The answer was “No.”

> A special effort is underway to improve the timeliness of Engineering Drawings. An additional special effort is needed to significantly improve drawing quality. This Board establishes an Engineering Quality Work Group (EQWG) to lead Engineering to a breakthrough level of quality for the future. To be competitive, our company must greatly improve productivity. Engineering should make major contributions to the improvement. The simplest is to reduce drawing errors, which result in the AIR (After Initial Release) change traffic that slows down the efficiency of the manufacturing and procurement process. Bigger challenges are to help make CAD/CAM a universal way of doing business within the company, effective use of group classification technology, and teamwork with Manufacturing and suppliers to develop and implement truly innovative design concepts that lead to quality products at lower cost. The EQWG is expected to develop ‘end state’ concepts and implementation plans for changes of organization, operation, procedures, standards and design concepts to guide our future growth. The target of the EQWG is breakthrough in performance, not just ‘work harder’. The group will phase their conceptualizing and recommendations to be effective in the long term and to influence the large number of drawings now being produced by Group 1 and Group 2 design teams.

My critical review of the above draft:

1. It does not have a clear ‘structure’, which would enable the reader to understand it.
2. The objectives (for example, cost savings) are not clearly stated (no numeric targets, when? scope?).

3. Undefended and unjustified assumptions are made, prejudicing the work of the group.
4. No reference to any of their own past, present or competitive efforts in this area (no benchmarks).

The first thing I do when presented with a document such as this is to go through and mark the ideas concerning performance requirements – the objectives (**bold and underlined**) and the ideas concerning strategies or solutions (*italics and underlined*). I also underline implied requirements.

(*Hint: You can use underlining and the letters 'O' and 'S' on a paper copy of a document.*)

> A special effort is underway to **improve the timeliness** of Engineering Drawings. An additional special effort is needed to significantly **improve drawing quality**. This *Board establishes an Engineering Quality Work Group (EQWG)* to lead Engineering to a *breakthrough level of quality* for the future. *To be competitive*, our company must **greatly improve productivity**. Engineering should make major contributions to the improvement. The simplest is to **reduce drawing errors**, which result in the AIR (After Initial Release) change traffic **that slows down the efficiency** of the manufacturing and procurement process. Bigger challenges are to help make *CAD/CAM a universal way of doing business* within the company, effective use of *group classification technology*, and *teamwork with Manufacturing and suppliers* to develop and implement *truly innovative design concepts* that lead to *quality products at lower cost*. The EQWG is expected to develop 'end state' concepts and implementation plans for *changes of organization, operation, procedures, standards and design concepts* to guide our future growth. The target of the EQWG is *breakthrough in performance* not just 'work harder'. The group will phase their conceptualizing and recommendations to be effective in the *long term* and to influence the large number of drawings now being produced by Group 1 and Group 2 design teams.

A framework for the requirements can then be drawn up for further work. Fuzzy brackets denote where more information is required.

Scope:
Time:
    <short term>
    <long term>
Place [Organizational Group]:
    Engineering Organization: Research and Development
        Group 1 Design Team
        Group 2 Design Team
    Manufacturing
        Procurement
    Suppliers

Performance Requirements:
Ambition: <competitive> + <breakthrough level of quality>.
Reduce Product Cost.
Improve Productivity [Engineering].
    Improve timeliness of <engineering drawings>.
    Improve <drawing quality>.
    Reduce <drawing errors>.
    Others.
Reduce <Engineering Process> timescales ('time to market').
Improve <Efficiency> [Manufacturing, Procurement].
Achieve <Growth>.
Others

This is just the start – there are no benchmarks or numeric values specified! Note also, that these are just initial lists; they are not in Planguage format. (See later chapters for discussion on how to specify such requirements.)

Some of the suggested potential design ideas are listed below. These design ideas are not requirements unless they are specific design constraints. Further work is required to establish how they should be viewed.

Potential Design Ideas:

- Have a team responsible for improvement – EQWG
- <Innovative> change of organizational, operation, procedures, standards and design concepts
- Make CAD/CAM a universal way of doing business
- <Effective> use of group classification technology
- <Effective> teamwork with Manufacturing
- <Effective> teamwork with Suppliers

Below is a clearer way to express the same ideas (but not necessarily the best way), which begins to address the issues of numeric values and evolutionary progress towards solutions. Some values are deliberately 'set up' with the aim that any wildly incorrect values will be challenged.

Ambition: As our primary initial task, we have targeted a significant reduction in the drawing errors, which are not due to customer change requests. When we have shown we can achieve that, other tasks await us.
The long-range objective is a reduction of drawing errors, which require After Initial Release changes. The aim is for errors to be dropped by at least 20% each year, from the levels current at the beginning of the year. Results are expected from the end of

November. Results should be designed to come in at the rate of 10% of annual target each month (that is, a 2% reduction of drawing errors each month). When results are not achieved, the EQWG will analyze the attempt and advise the Programs on possible improvements to achieve results.
Plan: The first month is November. An attempt to get a 2% reduction by the end of that month is the implied target.
Strategies: The Group 1 and the Group 2 design teams will start in parallel and will have a friendly competition for reduction of the drawing errors.
The design teams are expected to find their own detailed solutions and strategies.
Funding: Up to $500,000 is available immediately for funding any activity necessary for achievement of this target {experiments, training, consultants, research trips}. In the long run, the project should be self-funding through savings.
Responsibility: The Program Directors (and their staff) are responsible for achieving targets. The EQWG is responsible for supporting the activity, by dispensing the funding, reviewing progress and assisting the responsible program managers with any resources they may need.
Method: The method for planning outlined in the 'Proposed standard for EQWG Organization' will be the basis for planning. It will be modified as required by the EQWG.

Note: As the above was intended for presentation to management, it was formatted as ordinary text (without identifying user-defined terms).

See more about this case study in Section 3.2.

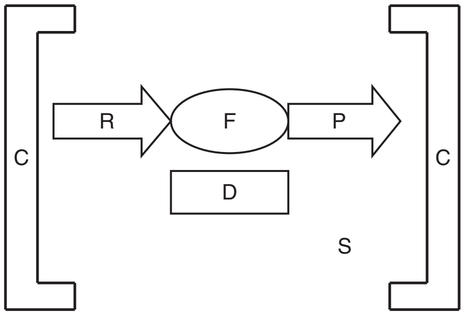
Bill of Rights

- You have a right to *know* precisely what is expected of you.
- You have a right to *clarify* things with colleagues, anywhere in the organization.
- You have a right to *initiate* clearer definitions of objectives and strategies.
- You have a right to get objectives presented in *measurable, quantified* formats.
- You have a right to *change* your objectives and strategies, for better performance.
- You have a right to *try out* new ideas for improving communication.
- You have a right to *fail* when trying, but must kill your failures quickly.
- You have a right to *challenge* constructively higher-level objectives and strategies.
- You have a right to be *judged* objectively on your performance against measurable objectives.
- You have a right to *offer* constructive help to colleagues to improve communication.

Original version in (Gilb 1988 Page 23)

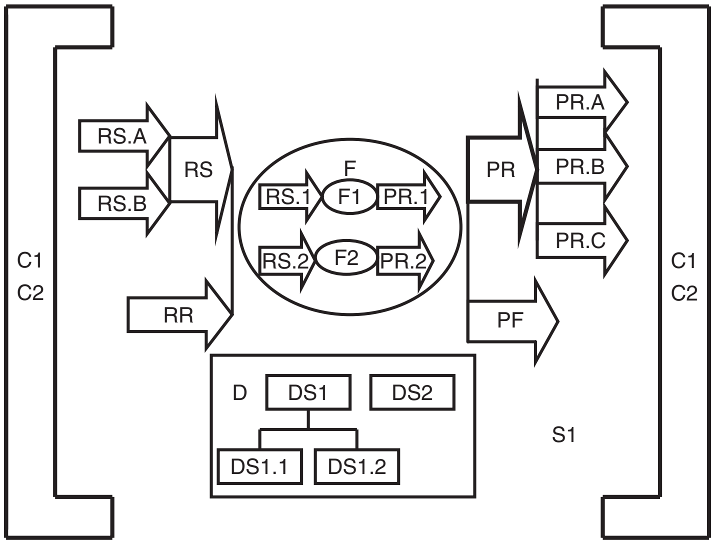
Figure 2.5
The author suggested these 'rights' for a multinational client. Of course it is a sneaky way to tell people what their 'duties' are!

2.9 Diagrams/Icons: Requirement Specification



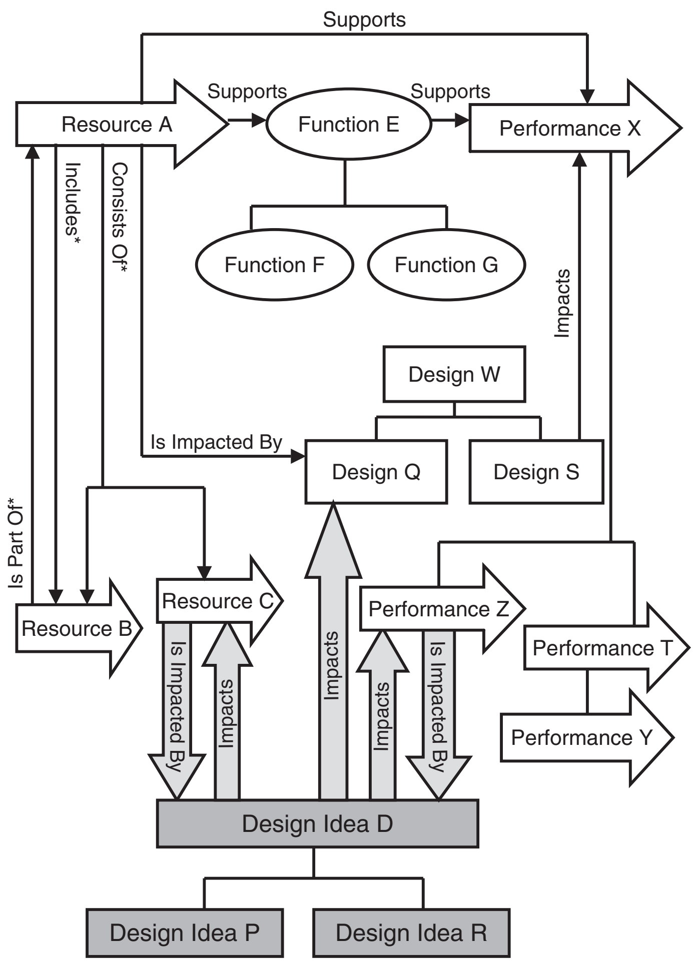
F is a function attribute. R is resource attribute, shown as an input attribute to F.
P is a performance attribute, shown as an output attribute to F.
D is a design attribute.
C represents a condition attribute (the two ‘brackets’ combined).
The set {R, F, P, D, C} make up a system, S.

Figure 2.6
A simple system model showing the main attributes for a system, S.



Notes:
F1 and F2 are sub-functions of function attribute, F.
RS is a complex cost. RS.1 and RS.2 are the corresponding resource attributes of their respective sub-functions F1 and F2. They can be referred to like this, F1.RS.1 or F.F2.RS.2.
RR is another resource attribute of F.
PR is a performance attribute of F.
F1.PR.1 and F2.PR.2 are the corresponding performance attributes at the sub-function level. PR.A, PR.B and PR.C are performance attributes (each has a separate scale definition). As a set, PR.A, PR.B and PR.C define the meaning of PR, a complex performance requirement.
PF is another performance attribute of F.
D is a design attribute of the system, S1. D has sub-components of DS1 and DS2.
C1 and C2 are condition attributes.

Figure 2.7
A more complex system model for a system, S1.



Note: * Hierarchical relationships are usually represented by lines rather than arrows.
Arrows are used here to explicitly show the direction of the relationship.

Notes:
Simplified icons are shown for resource and performance (in 'true' icons, the block arrows should each be linked to an oval, representing function).
The 'linking' terms include: Consists Of, Includes, Is Part Of, Impacts, Is Impacted By, Supports and Is Supported By. Note: Not all relationships are shown.

Hierarchical Links:
Performance X Consists Of {Performance Z, Performance T Consists Of Performance Y}.
Design W Consists Of {Design Q, Design S}.

Resource A Includes Resource B.
Function E Includes Function G.
Design W Includes Design Q.
Design Idea D Includes Design Idea R.
Performance X Includes Performance T.

Resource B Is Part Of Resource A.
Performance Z Is Part Of Performance X.
Function F Is Part Of Function E.
Design Idea R Is Part Of Design Idea D.

IE Table Links:
Design Idea D Impacts Design Q.
Design Idea D Impacts Resource C.
Design S Impacts Performance X.
Design Idea D Impacts Performance Z.

Resource A Is Impacted By Design Q.
Performance Z Is Impacted By Design Idea D.
Resource C Is Impacted By Design Idea D.

Specific Attribute Links:
Resource A Supports Function E.
Function E Supports Performance X.
Resource A Supports performance X.

Figure 2.8
Diagram showing how to express the relationships amongst attributes, between attribute and design idea, and amongst design ideas.

Requirement Specification Template (A Summary Template)

Tag: <Tag name for the system>.
Type: System.
========================== **Basic Information** ==========================
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Role/e-mail/name of the person responsible for changes and updates>.
Stakeholders: <Name any stakeholders (other than the Owner) with an interest in the system>.
Gist: <A brief description of the system>.
Description: <A full description of the system>.
Vision: <The overall aims and direction for the system>.
============================ **Relationships** ============================
Consists Of: Sub-System: <Tags for the immediate hierarchical sub-systems, if any, comprising this system>.
Linked To: <Other systems or programs that this system interfaces with>.
======================== **Function Requirements** ========================
Mission: <Mission statement or tag of the mission statement>.
Function Requirement:
<**{Function Target, Function Constraint}**>: <State tags of the function requirements>.
Note: 1. See Function Specification Template. 2. By default, 'Function Requirement' means 'Function Target'.
====================== **Performance Requirements** ======================
Performance Requirement:
<**{Quality, Resource Saving, Workload Capacity}**>: <State tags of the performance requirements>.
Note: See Scalar Requirement Template.
======================== **Resource Requirements** ========================
Resource Requirement:
<**{Financial Resource, Time Resource, Headcount Resource, others}**>: <State tags of the resource requirements>.
Note: See Scalar Requirement Template.
========================== **Design Constraints** ==========================
Design Constraint: <State tags of any relevant design constraints>.
Note: See Design Specification Template.
======================== **Condition Constraints** ========================
Condition Constraint: <State tags of any relevant condition constraints or specify a list of condition constraints>.
===================== **Priority and Risk Management** =====================
Rationale: <What are the reasons supporting these requirements? >.
Value: <State the overall stakeholder value associated with these requirements>.
Assumptions: <Any assumptions that have been made>.
Dependencies: <Using text or tags, name any major system dependencies>.
Risks: <List or refer to tags of any major risks that could cause delay or negative impacts to the achieving the requirements>.
Priority: <Are there any known overall priority requirements? >.
Issues: <Unresolved concerns or problems in the specification or the system>.
================= **Evolutionary Project Management Plan** =================
Evo Plan: <State the tag of the Evo Plan>.
======================= **Potential Design Ideas** =======================
Design Ideas: <State tags of any suggested design ideas for this system, which are not in the Evo Plan>.

Figure 2.9
Requirement specification template. This is a summary template giving an overview of the requirements.

2.10 Summary: Requirement Specification

This chapter has given an *overview* of requirement specification and introduced the different requirement types: function requirement, performance requirement, resource requirement, design constraint and condition constraint. Subsequent chapters (Chapters 3 to 6) will describe these requirement types in greater detail.

Planguage helps with requirement specification:

- by helping you to focus on the most critical requirements
- by demanding *numeric* definition for variable (scalar) requirements
- by making sure you obtain and specify benchmark levels for performance and resource attributes
- by encouraging specification of constraints.

As a result, the overall communication of the requirements between business management and systems engineering becomes much more precise:

- Technical staff of all levels have a clearer practical understanding of what they must deliver.
- Management can better understand and control project progress.

There are also two further, significant benefits from Planguage requirement specification:

- It actively assists the system design process. The numeric values of the benchmark and target requirements are direct inputs into Impact Estimation, which is used to quantitatively assess design ideas (*see Chapters 7 and 9*).
- It caters for *evolutionary system engineering methods* as it supports dynamic requirements and, it enables rapid, numeric tracking of progress. There is the ability to clearly specify how critical requirement levels should change over time and any changes to these numeric values (by project progress or change in requirement) are clearly visible to all. There is also the ability by specifying [time, place, event] conditions to readily communicate sub-division of a system.

Clear, specified requirements are at the heart of systems engineering. Planguage is a flexible tool to help you communicate requirements better.

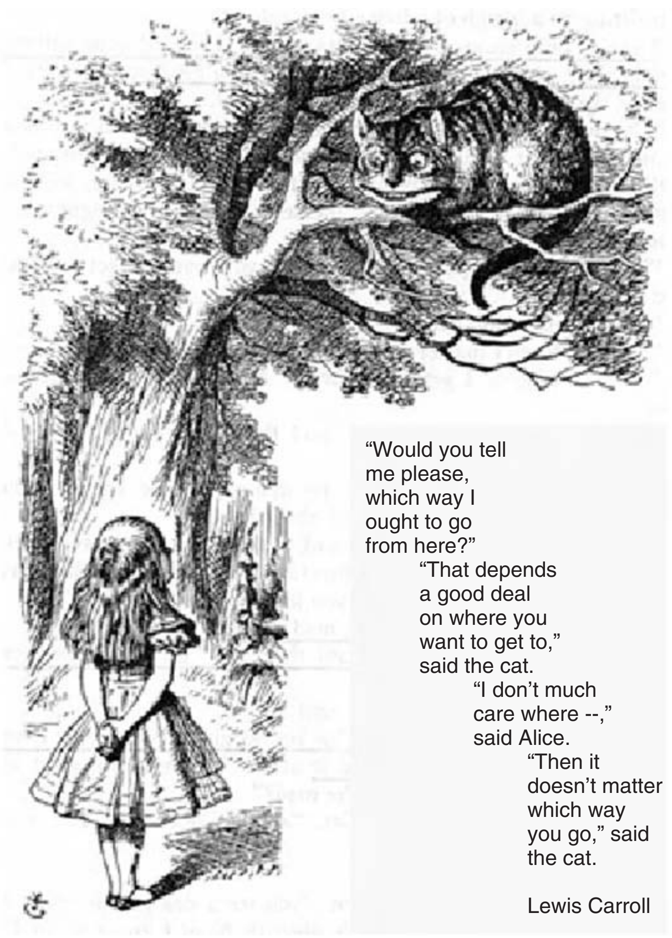


Figure 2.10
Alice and the Cheshire Cat. Illustration by John Tenniel, wood-engraving by Thomas Dalziel. From Chapter 6, *Alice in Wonderland* by Lewis Carroll.

Chapter
3

Functions

What systems 'do'

GLOSSARY CONCEPTS
- Function
- Function Requirement
- Mission

3.1 Introduction: Function and Function Requirement Specification

Function specifications define ‘*the business we are in*’. Functions are ‘*what*’ a system does. Functions must be distinguished from *how well* a system performs (the stakeholder performance attributes, such as quality) and from *how much* a system costs (the resources expended).

EXAMPLE Manual Dialing:
Type: Function Requirement.
Description: The user capability, by any available means {finger on key, voice, name list}, to select or provide and, transmit a {telephone or internet} {number or address} and any other required symbols, to reach and access any available services. It specifically includes any keying or other activity needed in connection with communication, such as accessing lists. It specifically excludes any non-communication activity such as game playing.

Separation of Functions from Design Ideas

Functions must also be distinguished from design ideas (how a system is going to achieve its requirements).[1] It is all too easy to mix them up but, if you do, you cheat yourself of the results you might get from a better design idea. The test is simple. Ask “Why this {function or design idea}?” If the answer is “because that is what our system ‘must have’ to be ‘our’ system at all,” you are probably talking about a function: something so fundamental that it is not for the systems engineer to modify or choose.

If the answer to “Why?” is “in order to get a performance improvement” or “for cost reduction,” then that specification is a *design idea*, not a function. For example, for a bank, ‘lending and dispensing money’ are clearly basic functions. The automated teller machines (ATMs) in the wall are clearly a ‘design idea’ from the bank’s point of view. This is because the ATM is one way to make the functions (of lending and dispensing money) have certain performance attributes (such as “to make it easier for our customers to withdraw money at the time they want to”).

Making this ‘function or design?’ distinction perhaps even more difficult, is the issue that the ‘objects’ we analyze are not purely ‘function’ or ‘design.’ Returning to the ATM example, at the ‘bank’

[1] Chapter 2 discussed how requirements must be separated from design in general. Now, here we are specifically discussing how function requirements must be clearly separated from design and explaining some of the associated issues.

level, the ATM functionality is a 'design idea'. However, at the level of the 'ATM project' the ATM functions become undisputed 'functionality'. The classification is dependent on your viewpoint.

In Planguage, we classify as 'function' or 'design' in order to convey the information about what is fundamental in a given situation (function) and what is a 'currently selected option' (a design idea). In this sense, we could say that the classification 'function' acts as a constraint[2] on the system designer. This distinction is one made in our minds, because we want the designer to have, or not to have, freedom to improve things. The 'system itself' is unaware of the distinction. An outside observer might not be able to see the distinction by merely looking at the system. For example, is air-conditioning in a car a function or a design? Is it an option or a fundamental concept? It all depends on the attitude of the people involved.

To give another example: at one stage the concept of putting a 'motor' into a horse-drawn carriage (creating the auto-mobile, the horse-less carriage) was clearly a 'design' intended to give certain performance and cost attributes, which 'horses' did not have. At this present stage of culture, the 'mechanical engine' in a car is taken almost completely for granted and has become a function, 'providing mechanical engine power.' This function clearly requires design ideas to implement it, which contribute to the overall characteristics of the car (some engine fuel design options are gasoline, diesel, steam, electricity and nuclear power).

With 'functions' you are not empowered to change them. You can't decide that a car will have no wheels; 'wheel functionality' is too much of a 'given' function. However, you can decide about many features of the design of the wheels, to ensure they have interesting attributes. You can also decide about the design of the 'motor' function, to give both it, and consequently the car, better attributes. But you cannot suddenly change the 'motor' function and opt for the horse again!

One advantage of making the design/function distinction clear is that if new design ideas come along (which could replace current design), you are psychologically ready to evaluate them, and accept the ones that on balance are better than the current design.[3] Another advantage is that you are more likely actively to look for alternative designs. In overall effect, the design/function distinction can free us up to design systems more competitively.

[2] Of course, it is only a 'true' requirement constraint if declared as a 'function constraint.' See later discussion in Chapter 7 on Priority Determination.

[3] *How* we estimate the relative contribution to requirement satisfaction of design ideas (their 'impact') relies on the methods in the following chapters: the quantification of attributes, and the estimation of the impact on these attributes, of the design ideas.

Keep uppermost in your mind that this classification process is simply about *giving information* to systems engineers about what they should take as 'givens', and about what they should 'engineer.' Any engineer, who has a true (engineering or systems architecture) design process, should care about this distinction: the information about what they *should* design is of crucial importance to them.

Classification as either 'function' or 'design' depends on:

- the circumstances:

  A 'selected design' or 'design constraint' becomes viewed as providing 'required functionality' as seen from later and lower levels of the decision-making hierarchy.

- the stage of development:

  One stakeholder's design idea becomes another development process person's 'required function'.

- the current culture:

  Yesterday's design may become today's 'given' function.

- the intent of the specification:

  If it is specified in order to deliver performance or savings requirements, it is a design.
  If it is there because it is 'fundamental', 'because that is how we do things,' then it is a function.

- the degree of freedom of a given type of planner/designer/architect to actually *change* the specification:

  If they are free to change it, then it is more likely design.

The above are some, hopefully useful, ideas to help you classify a specification as a function or a design. But, do not get over fixated by this process. It is finally one of degree and subjective judgment. A specification 'is what it is specified to be' – no matter how we classify it. The classification is intended to give us better ideas of our responsibilities for the specification and our options (Must implement as it is? Or, OK to improve it?).

Function Requirements

Any required function, which is essential and fundamental to the future system, is called a 'function requirement'. It must be specified as 'pure' function and it must be specified with information about the conditions [time, place, event] under which the functionality exists (otherwise there is no actual requirement!).

EXAMPLE Type: Function Requirement: {F1, F2, F3}.
F1 [USA]: 911 Emergency Dialling Capability.
F2 [Finland]: Character Capability {Finnish, Swedish, English}.
F3 [End Next Year, California]: Exhaust Emission Testing.

In addition, any instance of a real-world function always comes attached with a set of resource and performance attributes. So when we specify a function requirement, we have to consider what has to be done about its associated attributes. All function requirements must respect the full set of performance and resource requirements, which apply to 'their level' of the system.

EXAMPLE Availability.Q1 [F1, F2, F3]:
Type: Quality Requirement.
Scale: % Uptime. Fail: 99.0%. Goal: 99.5%.

EXAMPLE Availability.Q1:
Type: Quality Requirement.
Scale: % Uptime.
Fail [F1]: 90%, [F2]: 92.5%, [F3]: 95%.
Goal [F1, F2, F3]: 99.5%.
An example of how to specify the specific attachment of performance levels to functions. Availability Q1 is 'attached' to the three named functions, F1, F2 and F3 using qualifiers. In the first instance all goals are attached to the three functions. In the second instance only the one Goal is attached to the three functions and the Fail levels are attached individually and differently.

Any global scope requirements automatically apply to a function or sub-function, unless they are specifically contradicted by more specific local requirements.

Of course, it may be the case that certain key functions may require even higher performance levels (say for reliability and efficiency) than other functions. In these specific cases, the definition of the function requirement must *explicitly be linked* to appropriate specific requirements.

EXAMPLE Reliability: Scale: MTBF. Goal [Function X]: 99.98%.

Service Performance: Scale: Time in seconds to <reply to inquiries>.
Goal [Function: FX]: 1 second/reply.
Explicitly linking an attribute to a function.

By the time a function requirement (or part of a function requirement) is planned for delivery in an Evo step, its performance and resource requirements, and the conditions surrounding its delivery should be precisely pinned down using specification parameters like Risks, Is Impacted By, Dependency and Authority.

All function requirements will, ultimately, have a set of performance and resource attributes associated with them.
Systems engineering is about getting control over these attributes.

3.2 Practical Example: Function Analysis

Consider the (real) proposal to the Board of Directors asking for $60 million, which we first considered in Section 2.8:

Proposal to the Board of Directors

A special effort is underway to improve the timeliness of Engineering Drawings. An additional special effort is needed to significantly improve drawing quality. This Board establishes an Engineering Quality Work Group (EQWG) to lead Engineering to a breakthrough level of quality for the future. To be competitive, our company must greatly improve productivity. Engineering should make major contributions to the improvement. The simplest is to reduce drawing errors, which result in the AIR (After Initial Release) change traffic that slows down the efficiency of the manufacturing and procurement process. Bigger challenges are to help make CAD/CAM a universal way of doing business within the company, effective use of group classification technology, and teamwork with Manufacturing and suppliers to develop and implement truly innovative design concepts that lead to quality products at lower cost.

The EQWG is expected to develop 'end state' concepts and implementation plans for changes of organization, operation, procedures, standards and design concepts to guide our future growth. The target of the EQWG is breakthrough in performance not just 'work harder.' The group will phase their conceptualizing and recommendations to be effective in the long term and to influence the large number of drawings now being produced by Group 1 and Group 2 design teams.

Now let's further *analyze* it. Who are the stakeholders? What are the functions?

Stakeholders:

Management (Engineering Manager, Board of Directors and others)

Engineering Design Teams

- EQWG
- Group 1 Design Team
- Group 2 Design Team

Procurement
Manufacturing
Suppliers
Customers

Functions:

Carry out Research and Development
Create Designs

- Produce Engineering Drawings

Procure Materials
Manufacture Goods
Establish Work Environment
Maintain Work Standards and Practices
Maintain Organizational Structures

Note: These lists of stakeholders and functions show an alternative simpler formatting for Planguage sets (Parenthesis brackets '{ }' and commas are dispensed with).

As the functions become 'lower level,' they begin to constrain the design options! Great care must be taken that function specification is not taken down too far to the wrong level of decomposition. For example, 'Produce Engineering Drawings' is possibly beginning to dictate certain aspects of the solution.

Notice that by separating the different concepts of functions, design ideas, performance, resources and stakeholders, you get much greater clarity about what is really being said. The basis for further system improvement is also laid. For example, the performance attributes should next be taken a stage further, and be given better definitions that include numeric values stating the requirement levels. You are then able to start evaluating the 'impact of the proposed design ideas' on 'all the requirements.'

> Now you try! Take some recent and important system requirements from your own work, and analyze it into these components: {Function Requirements, Performance Requirements {Qualities, Workload Capacities, Resource Savings}, Resource Requirements, Design Constraints, Condition Constraints, Design Ideas, Assumptions and Comments}.

3.3 Language Core: Function and Function Requirement Specification

Here are some formats for referencing and specifying functions, including structuring them. If this seems more than you need for the moment, then all you *really* have to know is the basic format, *'Function Tag: <function description>.'*

Note: Function specification is not always for function *requirements*. You need to specify functions for *other purposes* as well, such as describing *existing* systems and clarifying functional concepts.

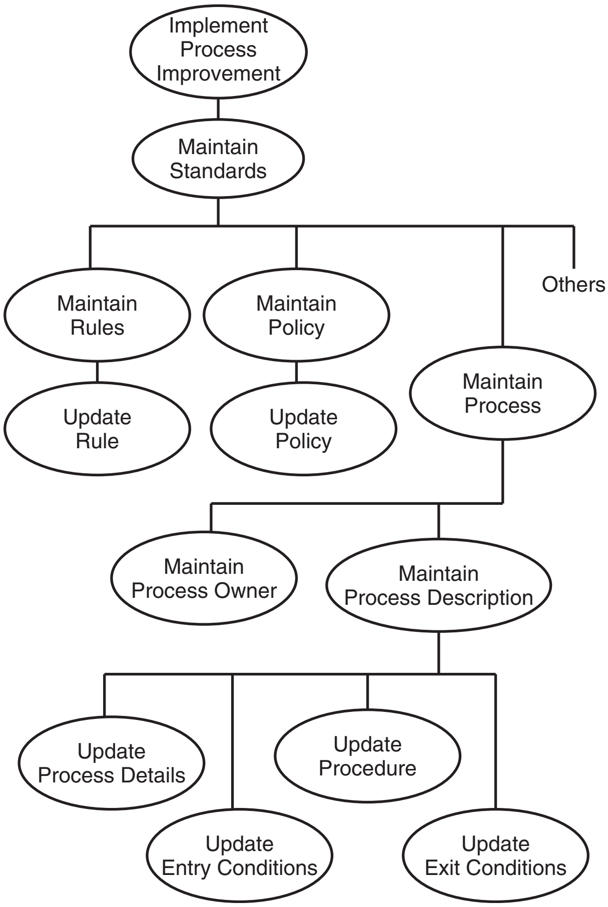


Figure 3.1
Diagram showing the relationships amongst the functions used in the examples in this section.

Referencing Functions

Functions are identified with a tag. Some variations on the tag structure are given here below. You can use the format: 'Parent Tag.Kid Tag' or, if it is 'unique in context,' just 'Kid Tag.' The first/left tag indicates a parent function, and the following tag indicates a kid/child function.

EXAMPLE Tag: Maintain Standards.Maintain Rules.
Tag: Maintain Rules.Update Rule. "or just Update Rule if it is not ambiguous."

EXAMPLE *You can also use the format:*
Maintain Standards: Includes: Maintain Rules.
Maintain Standards: Includes: {Maintain Policy, Maintain Rules, Maintain Process}.
Or
Maintain Standards: Includes: Maintain Rules: Includes: Update Rule.
The latter example is explicit about the hierarchy.
Note: '{. . .}' is the Planguage symbol for a 'set' of things.

Specifying an Arbitrary Set of Functions

There is no implication when specifying functions and functional relationships, that all siblings are specified or that the functions listed are even direct descendants of the same parent. Any set of functions can be given a common collective tag for reference:

Arbitrary Function Set: Type: Function {Function Tag 1, Function Tag 2, . . . , Function Tag N}.

EXAMPLE Maintain Standards:
Type: Function.
Defined As: {Maintain Rules, Maintain Policy, Maintain Process, Others}.
Or more briefly:
Maintain Standards:
Function: {Maintain Rules, Maintain Policy, Maintain Process}.
Note: use of the parameters 'Type' and 'Defined As' are optional. It is a matter of style and readability.

Inheritance of Higher Level Requirements

A function will automatically inherit any relevant specifications' parameters from relevant higher system levels. This includes higher-level (system-wide) performance requirements, budgets and any condition constraints. These inherited parameters apply by implication, unless there are other parameters that specifically override them in more local function specifications.

Function Specification

A function specification defines all the currently interesting functional aspects. It optionally includes the function description, functional relationships (that is, the names of relevant supra-functions, sub-functions and sibling functions), associated specific performance and resource attributes, condition information [time, place, event], source information, risks and assumptions, as well as many other parameters. It even includes the implied or 'default' attributes' properties inherited from higher levels of the system of which the function is a member!

Here is an example of a client's function specification (edited for confidentiality):

EXAMPLE

Emergency Stop:
Type: Function.
Description: <Requirement detail>.
Module Name: GEX.F124.
Users: {Machine Operator, Run Planner}.
Assumptions: The User Handbook describes this in detail for all <User Types>.
User Handbook: Section 1.3.5 [Version 1.0].
Planned Implemented: Early Next Year, Before Release 1.0.
Latest Implementation: Version 2.1. "Bug Correction: Bug XYZ."
Test: FT.Emergency Stop.
Test [System]: {FS.Normal Start, FS.Emergency Stop}.
Hardware Components: {Emergency Stop Button, Others}.
Owner: Carla.

The main parameters for function specification are described in the following paragraphs.

Function Description

A function description describes *only* the action(s) of the function.

Function Tag 1: Function: <function description> <-Source.

EXAMPLE

Refugee Transport: Moving refugees back to home villages. <- Charity Aid Manual.
"The mode of transport will be determined by safety, and cost factors."

A more explicit 'parameter-driven' format may also be used for clarity:

EXAMPLE

Tag: Refugee Transport.
Type: Function Requirement.
Description: Moving refugees back to home villages.
Source: Charity Aid Manual [Version = Last Year].
Dependency: The mode of transport will be determined by safety and cost factors.

The choice of more or less formality is governed by factors such as size of plan, size and type of readership, familiarity with Planguage and stage of planning (for example, 'drafting ideas' or 'making a formal plan').

Functional Relationships

Functional relationships are used to define the relationships amongst functions. For a specific function, the different kinds of relationship include:

- **Sub-functions**: These are any lower-level functions that comprise a function. Any sub-functions at the *immediate lower level* to the specific function are known as **Kid** (Child) functions.
- **Supra-functions**: These are any higher level functions, which the specific function forms a part of (is 'sub-function' of). The immediate supra-functions of a function are called the **Parent** functions. The ultimate, hierarchical top level function, within an organization or project, is usually called a '**mission.**'
- **Sibling functions**: These are any functions sharing at least one parent function with another 'sibling' function.

Here are some examples of specifying functional relationships (see Figure 3.1):

Defining Supra-functions (as a set of functions)

Supra-functions: Function {Function Tag 1, Function Tag 2, . . . , Function Tag N}.

EXAMPLE Tag: Update Rule.
Type: Function.
Supra-functions:
Function: {Maintain Rules, Maintain Standards, Implement Process Improvement}.

Referencing Supra-functions for a Function

A hierarchy of tags can be used to show the function hierarchy. You can use **bold** or <u>underline</u> to emphasize which tag you are focusing on. The non-emphasized part is the information about the supra-function ancestry or genealogy.

A hierarchy of tags is connected by dots: 'Tag 1.Tag 2.Tag 3'.

EXAMPLE Maintain Standards.Maintain Rules.**Update Rule**: Type: Function.

Defining Sibling Functions

The format examples below define siblings.

EXAMPLE Some Kids: Includes: {Kid1, Kid2}.

EXAMPLE All Kids: Consists Of: {Kid1, Kid2, Kid3, Kid Last}.

Attributes of a Function

Attributes of a function are any specific performance or resource attributes specified in the function definition. They include past benchmarks {Past, Record, Trend} describing a function's current or historic attributes and, if a function *requirement* is being specified, they also include future target requirements {Goal, Budget, Stretch, Wish} and constraints {Fail, Survival}. Qualifiers must be used in those attribute specifications to define the specific instances of the past or future use of the function.

EXAMPLE Goal [End Dec Next Year]: 22,000.

Attributes of a function can be described and directly connected to the function in the following way:

TEMPLATE <Function Tag 1>:
Type: Function.
Description: <describe the function here, well enough to allow testing of it>.
Attribute 1: Scale: <?> Goal: <?>.
(Attribute 2: Scale: <?> Budget: <?>.)

Template for specifying the attributes of a Function.

Note: Fuzzy brackets, '< . . . >' are used in a template to indicate what to 'fill in'. The fuzzy brackets may contain some instruction, which will always be wiped out when the brackets are filled in. The parenthesis, '(. . .)' are used to indicate (optional) specification types.

EXAMPLE Flagship Product:
Type: Function.
Description: Provide a mobile telephone service [Product Code 9998].
Reliability: Scale: Mean Hours between Faults. Goal [End Dec Next Year]: 22,000.
Battery Life: Scale: Hours Life. Goal [Standby]: 500, [Calling]: 50.
Function with specific local performance requirements. The specification of function, Flagship Product contains explicit and local definition of two performance attributes. Other attributes and specifications may be implied by other specifications elsewhere.

Qualifiers

Qualifiers can be used to specify the set of conditions [time, place, event] applying to a specific function.

Qualifiers can also be applied to functions in the following way:

TEMPLATE <Function Tag 1>:
Type: Function Requirement.
Qualifier: [time condition, place condition, event condition].
Description: <Define the function here>.
Template for a function with conditions. Note: the function is only 'required' or 'valid' when all elements of the qualifier are 'true'.

EXAMPLE Installation:
Type: Function Requirement.
Qualifier: [Next Year, Activity = Emergency Repair, Major Cities].
Description: Any job <our installers> must perform.

Any useful set of qualifiers is valid.

3.4 Rules: Function and Function Requirement Specification

Gist: Specific Rules for specification of Functions and Function Requirements.

Tag: Rules.FR.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Base: The rules for generic specification, Rules.GS and the rules for requirement specification, Rules.RS apply.

R1: **Functionality**: Function requirements will specify what the system must do and all specified functionality must be required by specified stakeholders (Type: Function Requirement).

Function requirements are not themselves 'unconditionally required.' Their actual implementation will depend on their relative priority – as specified by qualifiers and other parameters (such as 'Authority').

R2: **Detail**: The function requirement specification should be specified in *enough detail* so that we know precisely what is expected, and do not, and cannot, inadvertently assume or include function requirements, which are not actually intended. It should be 'foolproof'.

Detailed definition within sub-functions can satisfy this need for clarity, the higher level function does not need to hold all the information.

EXAMPLE Fuzzy Function Environment:
Gist: Ensuring Environmental Considerations.
This is a defective specification, given Rule R2. A more detailed function definition is given in the following example.

EXAMPLE Ensuring Environmental Considerations:
Type: Function Requirement.
Description: All legally and competitively necessary functionality, immediate and potential, regarding environmental protection, in the widest interpretation possible, to protect us against lawsuits, and give us a clear positive reputation amongst consumers.

R3: **Not Degrees**: Elementary function specifications must *not* be described in terms of degrees or variability.

Elementary functions are binary (present or absent in totality) in nature. If something is 'variable in degrees,' then it probably needs to be reclassified, and redefined as a performance or resource specification linked to a function.

R4: **Not Design**: The specified 'function' requirement *must not be* a design idea (for example, a strategy, a device, a method or a process) whose only or main justification is to satisfy a performance or resource requirement of the system.

If the 'function specification' is really a design idea, then it shall be re-classified as 'Type: Design Idea'. If it was intended to support yet undefined performance or resource requirements (like Design X Impacts Performance Y), then action will be taken to properly define these attribute requirements. Such action might justify rewriting the so-called 'function' as a design specification, as there is now at least one requirement that the design idea can impact.

We must avoid 'false' function requirements, which are really just designs, which someone assumes would be good for meeting unspecified and unofficial performance requirements. (Local version of Rules.RS.R9: Design Separation.)[4]

[4] This rule intentionally duplicates RS.R9 as it is considered so important for functions. Whenever such duplication occurs, specific reference should be made to the rule being duplicated.

EXAMPLE Usability: "An example that violates R4 as the Type classification is incorrect! The Description also has errors."
Type: Function Requirement.
Description: A state-of-the-art, user-friendly interface.
'Usability' is a performance attribute, and needs definition (using a Scale and other parameters, such as Goal). If your intuition doesn't tell you this, then 'state of the art' is a clue as to 'variability' or 'degree of goodness.'
'Interface' is a 'thing to be designed' in order to achieve various attributes, including, but not limited to, 'Usability.' Specify this interface amongst the 'design ideas.' It is not a 'what,' but it is a 'how' (a design idea).

R5: **Function Priority**: If there is a required simple priority for a function requirement, then it should be explicitly stated with information about its authority and/or the source reference and the reason for the priority.

Use the Priority parameter 'Priority: After Y' or use suitable qualifiers '[Before X].' Use the Authority, Source and Rationale parameters to specify the supporting information.

EXAMPLE Rationale: We must address Service Level Agreements as soon as possible to enable the correct level of support to be given when a customer phones with a problem. That is where we are incurring too much cost and tying up engineering support resources. <- Customer Services Director.
See also Section 7.7, which discusses Priority Determination.

R6: **Testable**: A function *must* be specified, so that it is *possible* to define an unambiguous test, to prove that it is later implemented *(Local version of Rules.RS.R8: Testable).*

R7: **Test**: Any notions of how or what needs to be tested, in order to validate a function *may* be described using the Planguage parameter 'Test,' with the function name as the qualifier.

The Test information is either specified with the function definition or as a separate item.

EXAMPLE Function Y:
Type: Function Requirement.[5]
Description: Charging to Accounts.
Test [Function Y]: Tests shall be developed to demonstrate that this function is available for all counties in this state, and prove that no other states or countries can access it.

[5] Note: By default, a 'Function Requirement' is assumed to be a 'Function Target'.

Audit: Test [Function Requirement: Function Y]: We must demonstrate to internal auditors that no counties, which are <financially insolvent> are allowed access to this function <- Audit Report [August This Year].

3.5 Process Description: Function Requirement Specification

Process: Function Requirements

Gist: A process for specification of function requirements.

Tag: Process.FR.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Entry Conditions

E1: The Generic Entry Conditions apply.

E2: You have the ability to observe comparable 'real' systems (see P3, below).

Procedure

P1: Describe the hierarchical structures of the high-level function(s), as sets of related function and sub-function tags (for example, F1.F2.F3).

P2: For each function tag (this also includes tags for sub-functions and supra-functions as relevant), define the function, in the detail required by the rules for function requirement specification (see Rules.FR in Section 3.4).
Note: Focus on real functionality ('what it does') and exclude any design ideas intended to satisfy performance and resource requirements.

P3: Where relevant: sample comparable 'real systems' to check the accuracy of the function specifications. Correct the specifications as necessary.

P4: Check accuracy and completeness of function requirements, with the people who are currently using similar existing systems. Correct the specifications as necessary.

P5: Perform Specification Quality Control (SQC) on the draft function specifications. Check the quality level against the required quality level, as specified by the exit conditions (see X1). If SQC fails, rewrite/

correct the function specifications (that is, revisit P1 to P4). Continue P5 until the appropriate quality level is reached.

P6: Once the function specifications have exited SQC, get real current system users (if any) to sign off agreement to them.

P7: Repeat any procedure above until the exit conditions can be satisfied.

Exit Conditions

X1: The Generic Exit Conditions apply.
By default, no more than 0.2 remaining major defects/page are allowed in any of the function specifications. (A page is 300 words of non-commentary text.)

X2: The relevant system users, if any, must have signed off the function specification.

Simplified Function Requirement Specification Process

Process: Function Requirement Simplified.
Gist: An alternative simplified variation for Function Requirement Specification.
Tag: Process.FRS.
Version: October 7, 2004. Owner: TG. Status: Draft.

Entry Conditions
None.

Procedure
P1: Declare a specific, 'already specified' and 'currently operational' system to be the 'living map' of the function requirements.

There is usually an old existing system of some kind. It is likely that a future system must replace this old system, in order for the business or organization to remain viable in the future.

Where relevant, use [qualifiers] to aid the mapping of the old to the new.

The function specification detail is then continuously observable 'in the real system.' It should only be analyzed 'as needed.' Exit immediately.

Exit Conditions
None.

Note: This method is useful when doing evolutionary delivery of changes to impact performance/resource attributes and minor changes to real functionality for an existing large system. You focus on 'improvement results,' not 'supporting functionality.' I normally apply this method to most real projects I get involved in (TG).

3.6 Principles: Function and Function Requirement Specifications

These are principles for recognizing what is, and is not, a function and also for working with functions.

1. **The Principle of 'What Function?'**
   Function is 'what' a system does, *never* 'how well' it does it or 'how it does it so well.'
2. **The Principle of 'Thing with Attributes'**
   A function is the thing, which has the performance and resource attributes attached to it.
3. **The Principle of 'Living Map'**
   Function specification is sometimes best done by declaring the existing system to be a living map.
4. **The Principle of 'Part of Totality'**
   Functions are always part of some larger function and can probably be described by their own sub-functions.
5. **The Principle of 'Each to their Own'**
   Different functions require different performance and resource attributes; so, one reason we specify the functions is to identify and distinguish their required attributes.
6. **The Principle of 'Timing'**
   Different functions can be delivered to customers at different times, so another reason to specify functions is to know 'what to do when.'
7. **The Principle of 'Conditional Function'**
   Some functions may not be necessary, *except* under specified conditions or events, and these conditions should be specified and exploited in project planning. You don't have to do what is not *yet* required!
8. **The Principle of 'Room with a View'**
   A function definition is not absolute; it is a viewpoint, and many overlapping function views can be made and used fruitfully to satisfy different needs.
9. **The Principle of 'Terrain does not change with the Maps'**
   The real system does not change just because you document function viewpoints and function hierarchies: correctly or incorrectly.
10. **The Principle of 'False Function Foils Fruits'**
    If you mistakenly request a design, as basic functionality, you will limit your ability to improve the design to give better competitive attributes.

Alternatively,
Don't request 'functions' which are really 'designs for performance',
You might not get the performance you really want.

3.7 Additional Ideas: Function and Function Requirement Specification

Mission

As mentioned earlier in this chapter, the ultimate top-level function of any system is termed its '*mission*.' A mission describes 'what' a specific system does. Many organizations have explicit mission statements.

We could just as well call 'mission' the 'top-level function.' But the concept and term 'mission' is well known, and for many purposes works better than 'function.' For example, 'The mission of this project is Mars Exploration' sounds better with 'mission' rather than 'function.' Keep in mind that any 'mission' is still really a sub-function of some larger functional context.

Of course, a mission only provides a high-level description of a system's function. Further detail is provided by its sub-functions and by its associated performance and resource attributes. Also, to fully understand a system, we must have information about its environment. A system *interacts* with the environment in which it *operates*.

Note it is important that we not confuse 'mission' with 'vision.' A vision statement is a higher-level concept. It can include ideas about *how well* the mission will be conducted. For example, "Our vision for the 'Mars Mission' is to get back alive, with substantial new scientific knowledge."

Elementary and complex concepts

Functions and many other Planguage types can be described as being elementary or complex concepts. The meaning of these terms, regarding functions, is as follows:

- An *elementary* function is not decomposed into sub-functions. It may be the case that it is unable to be broken down any further or a deliberate decision may have been taken not to further decompose it.
- A *complex* function is composed of a set of at least two sub-functions. The set of sub-functions can be any mix of relevant complex and elementary functions. At the lowest level of functional decomposition a complex function is defined completely in terms of elementary functions.

EXAMPLE Planning a Project:
Type: Complex Function.
Includes: Elementary Function: {Reviewing Evo Step Feedback, Checking Requirement Specification is Valid, Selecting Next Evo Step, Allocating Staff to Evo Step}.
An example of elementary and complex functions.

Measuring Functionality

Functions are either 'present or absent;' they have a binary nature. They are either documented or observable (testable) in a real system or they are not.

Sets of complex functions can be thought of numerically as 'percentage amounts' of their 'defined lists' of elementary functions. *Elementary* functions are (by definition) not divided into sub-functions.

Complex functions are either 100% present (all elementary functions in the defined complex function set are present) or they do not 'wholly' exist. For a complex function called ABC, you can talk about, say, 90% of the set of elementary functions comprising ABC being defined or implemented. In such a situation ABC itself, the entire complex function, doesn't exist yet; only known degrees of it are defined or in place.

Additional Examples of Function Specification

Here are some Planguage ideas, additional to the ones shown in Section 3.3, which can be applied to function specification. They give more detail on the use of qualifiers.

EXAMPLE

F3 [F499]: Receiving e-mail from Customers.
F3 is a valid function if, and only if, F499 is active or in existence. F499 is a 'condition' (specified in the format of a [qualifier]). F499 is detailed 'elsewhere'. F3 is a complex function specification because it has a qualifier, which must be determined by the qualifier's own definition
SYSX. **F5**: Sending e-mail to defined [Group].
'F5' is a 'kid' element of the complex function SYSX. The actual function implementation will differ depending on the current definition of 'Group'.
F6 [Date = After First Release]: Get approval by electronic signature.
F6 is not 'valid' (for implementation, for testing) until after 'First Release' event has taken place. First Release is a specification variable, depending on the actual release date.
F7:
Qualifier: [Country = {European Union Countries, Norway, Not USA, Not Canada}].
Definition: Maintain System XYZ Standards.
The function, F7 is valid for a defined set of countries. The qualifier parameter, 'Country = ' illustrates how a more explicit format can provide better readability for Planguage novices.
F8:
Description: Answering direct-line telephone.
Speed: Scale: Number of <whole rings> heard at Receiving End, before Answer Signal is <sensed>.

Past [Condition = Employee Not At Desk]: 4.
Goal [Condition = Employee Not At Desk]: 1.
Answer: Scale: Probability that Caller is satisfied with the Given Answer.
Past [OK]: 50%, Goal [OK, Version 6]: 90%.
OK: Defined As: {Condition = Correct Employee, Hours = 0800 to 1700}.
In this example, the function, F8 is defined together *with multiple attributes, both benchmark and target. Notice the qualifier 'OK' is defined in a separate statement to make reuse of it easier. This also prevents repetition, saves space and saves time when making changes.*

3.8 Further Example/Case Study: Function Specification for an Airborne Command and Control System

This is an extract from the top-level function specification for a real system (The system is now operational and delivered to customers).

Note: Mapping functions in detail is not the prime intention when using Planguage. The aim is to establish an evolutionary plan, which focuses on result delivery to some defined system stakeholders. This aim does not necessarily require *any* 'delivery' of additional functionality! Delivering 'designs,' to just improve performance and resource attributes for existing functionality is quite common. The level of understanding of the functions needed at the planning stage is merely that required to support the system designers and others involved in the requirement specification process. Specifically, this means that a complete, in-depth description of all the system functions and processes is not required. I strongly recommend investigating functions in detail *only as required*, at the design stage of each evolutionary delivery step. (There may well be exceptions to this, but don't waste resources.)

Airborne Command and Control: **ACC**.
Type: System.
Includes: Type: Sub-system:
M: Mission.
P: Planning.
D: Data Handler.
C: Communications Intelligence.

ACC.D.MOP: ''MOP stands for 'Manual Operation(s)'.''
Type: Function.

Includes:	
BIT:	Provide 'Built In Test.'
DATAB:	Provide database diagnostics. <Various levels of checking>. Not including <on mission>.
DATAELEMENT:	Check <Data element reasonableness> when <on mission>.
DATALINK:	Interchange data links manually by operators. ''*Component from our mother company*.''
DIAG:	Display all faults to operator and log on file.
DISPLAY:	Display error and fault detection data to operator.
ESM:	Display error messages from communications/ non-communications system.
FMS:	Display any loss of data from Flight Management System.
HEARTBEAT:	Supervise computer node <status> by heart-beat <signaling>.
INIT:	Test data destructive HW when initializing the <system>.
LINK:	Display <status information> of the data links.
LOG:	<Save on file> fault detection data and detailed test information.
PRINTER:	Report <printer status> from the AX-BUS when any fault occurs.
RADAR:	Display loss of data from radar.

This was from the first draft of the function specification. Many concepts are marked with <fuzzy brackets> and require further work to precisely define them.

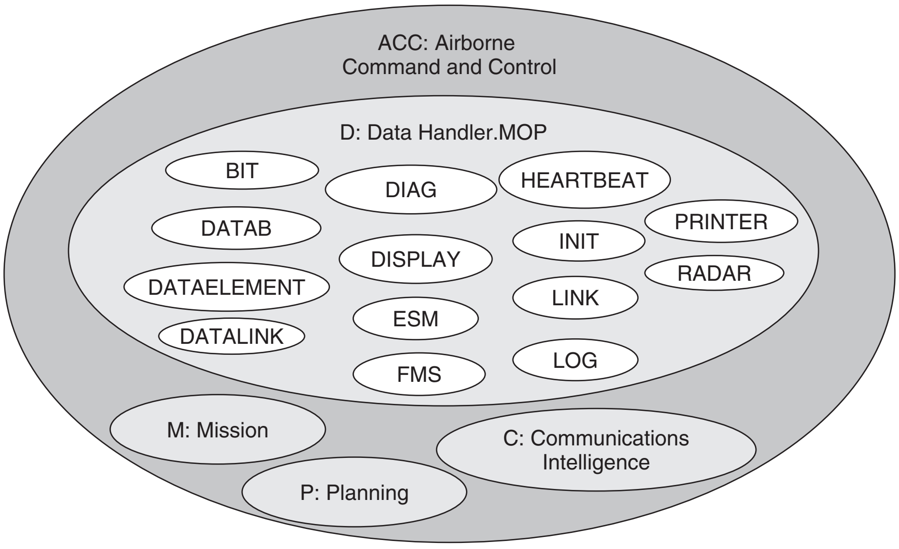


Figure 3.2
Functions within the Airborne Command and Control (ACC) System.

Here is an example of a function requirement:
(Notice the use of a Planguage template. The template parameters are given in ***bold****. See the next section for a more detailed specification of this template.)*

Example of a Function Requirement

Tag: DATADIAG:
Type: Function Requirement. ''Note, DATADIAG is not 'real', it is an example made up using the basic ideas named in the real case above. It is for teaching purposes only.''
Version: October 7, 2004 21:38.
Owner: Quality Assurance Division.
Implementer: Database Team.
Stakeholders: Quality Assurance Division, Maintenance Support.
Gist: Obtain Database Diagnostics.
Description:
S1: To monitor database quality. <Various levels of checking>.
S2: To report database diagnostics.
S3: To integrate with the automatic recovery system.
S4: To run in parallel with the operational use of the database as a background function.
S5: Monitoring operation to be optional. For example, to be off when <on mission>.
S6: Monitoring operation to be user-driven by parameters to enable selected sampling of specific classes of database records, data elements and relationships.
Supra-function: ACC.D.MOP. ''This is the specification of the supra-function of DATADIAG from some viewpoint.''
Sub-functions: None specified.
Supports: {System Recovery, Bug Maintenance, Database Integrity}.
Assumptions:
A1: This sub-system will not degrade operational database performance by more than 5%.
A2: It will be cheaper to automate this function than to do analysis manually.
A3: It will be faster and more reliable than manual checking.
Dependencies: D1: The database system itself must be defined and operational.
Risks: R1: Failure to update this function in parallel with the database structure.
Priority: This function must be available to some degree in first customer use releases. It will also be used in pre-release systems testing to some undefined degree.
Test: T1: This function shall be used in system testing and an early version of it can and should be made available in parallel with the development of the database itself. The function shall be tested by insertion of artificial database defects, and shall discover 100% of these.
Financial Budget: The cost of developing and maintaining this function is assumed to be between 10% and 50% of the cost of building and maintaining the database software in total.
Function Intranet Location: ACC. Software.DB-Diagnosis.

3.9 Diagrams/Icons: Function and Function Requirement Specification

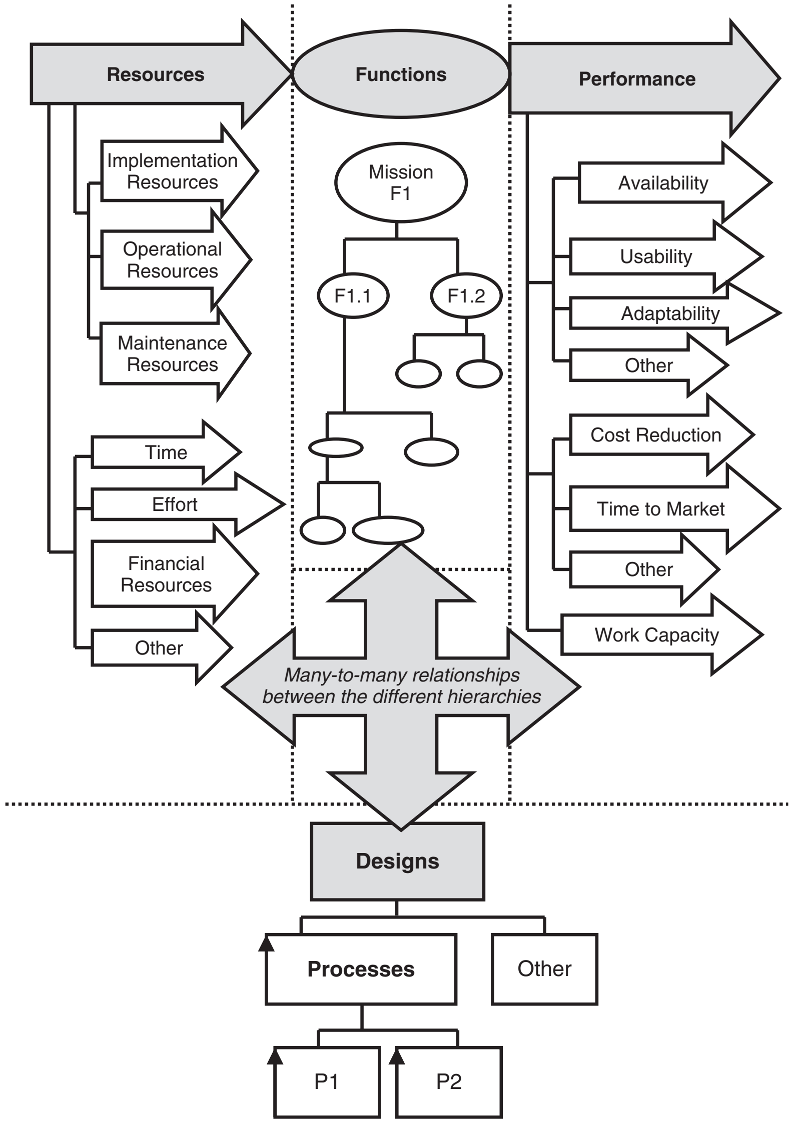


Figure 3.3
This shows the four main system attribute types: resource, function, performance and design. It also shows the processes, which implement the functions. Using Planguage, the complex relationships amongst these four different types can be specified. For example, a specific performance level might apply only to a handful of functions rather than the entire system, or a function might be implemented by several processes, or different resources can be specifically allocated to different functions.

Template for Function Specification <with hints>

Tag: <Tag name for the function>.
Type: <{Function Specification,
Function (Target) Requirement,[6]
Function Constraint}>.
========================== **Basic Information** ==========================
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Name the role/email/person responsible for changes and updates to this specification>.
Stakeholders: <Name any stakeholders with an interest in this specification>.
Gist: <Give a 5 to 20 word summary of the nature of this function>.
Description: <Give a detailed, unambiguous description of the function, or a tag reference to some place where it is detailed. Remember to include definitions of any local terms>.
============================ **Relationships** ============================
Supra-functions: <List tag of function/mission, which this function is a part of. A hierarchy of tags, such as A.B.C, is even more illuminating. Note: an alternative way of expressing supra-function is to use Is Part Of>.
Sub-functions: <List the tags of any immediate sub-functions (that is, the next level down), of this function. Note: alternative ways of expressing sub-functions are Includes and Consists Of>.
Is Impacted By: <List the tags of any design ideas or Evo steps delivering, or capable of delivering, this function. The actual function is NOT modified by the design idea, but its presence in the system is, or can be, altered in some way. This is an Impact Estimation table relationship>.
Linked To: <List names or tags of any other system specifications, which this one is related to intimately, in addition to the above specified hierarchical function relations and IE-related links. Note: an alternative way is to express such a relationship is to use Supports or Is Supported By, as appropriate>.
============================ **Measurement** ============================
Test: <Refer to tags of any test plan or/and test cases, which deal with this function>.

===================== **Priority and Risk Management** =====================
Rationale: < Justify the existence of this function. Why is this function necessary? >.
Value: <Name [Stakeholder, time, place, event>]: <Quantify, or express in words, the value claimed as a result of delivering the requirement>.
Assumptions: <Specify, or refer to tags of any assumptions in connection with this function, which could cause problems if they were not true, or later became invalid>.
Dependencies: <Using text or tags, name anything, which is dependent on this function in any significant way, or which this function itself, is dependent on in any significant way>.
Risks: <List or refer to tags of anything, which could cause malfunction, delay, or negative impacts on plans, requirements and expected results>.
Priority: <Name, using tags, any system elements, which this function can clearly be done *after* or must clearly be done *before*. Give any relevant reasons>.
Issues: <State any known issues>.
========================== **Specific Budgets** ==========================
Financial Budget: <Refer to the allocated money for planning and implementation (which includes test) of this function>.

Figure 3.4
A template for Function Specification.

[6] Note: By default, a 'Function Requirement' is assumed to be a 'Function Target'.

3.10 Summary: Function and Function Requirement Specification

Functions are '*what*' a system does. The concept of a pure 'function' does not include information about the function's performance attributes (how *well* a function is done); nor about the function's conditions [when, where, if]; nor about design ideas (*how*, a function achieves its attributes *at the required levels*).

My view of the discipline of functions is that they are 'boring, but essential, necessities.' They are the basics of the business or field you are dealing with, and probably exactly the same as those of your competitors in the same market.

The 'real competitive action' lies in identifying the interesting (competitive) performance and resource *attributes* for the functions, then establishing their required *competitive levels* and, then finding interesting ways (designs) to achieve them.

So, you can view functions as providing the framework 'supporting' the performance and resource attributes necessary for winning.

Any attempt to implement a function without trying to gain control over its performance and cost attributes, will result in unplanned, uncontrolled and thus probably undesired attributes. You must control attributes of functions to control the 'Risks.'

Many of the common problems, which systems engineers experience (such as deadline control, cost overruns and bad quality) are, in my view, significantly caused by:

- Specifying poorly-justified and insufficiently-detailed 'design' and calling it 'Function Requirements.'
- Articulating the performance and costs of functions in ways that can't be measured or tested.
- Focusing on testing functions alone, rather than the key stakeholder-value performance and cost attributes.

Functions are merely real-world reference points. They are not the interesting 'problem' for competitive systems engineering.

Chapter

4

PERFORMANCE

How Good?

GLOSSARY CONCEPTS

Performance
Quality
Resource Saving
Workload Capacity
Scale
Meter
Benchmark
Past
Record
Trend
Target
Goal/Budget
Stretch
Wish
Constraint
Fail
Survival

Consider the Performance of :

A flower

- fragrance
- attractiveness
- pollen quantity
- toxicity
- bloom frequency

A car

- comfort
- safety
- speed
- capacity

A person

- balance
- intelligence
- courtesy
- helpfulness

Figure 4.1
Some examples of performance concepts.

4.1 Introduction

Performance: Quality, Resource Savings and Workload Capacity

Performance describes the system benefits: how good the system is and how it affects the external world. Performance attributes state the actual and/or potential benefits and effects experienced by stakeholders in their environments.

Performance attributes are the output attributes; they state the effectiveness of a system. By contrast, the input (or ‘fuel’) attributes are the resources/costs of developing and/or maintaining a system that exhibits

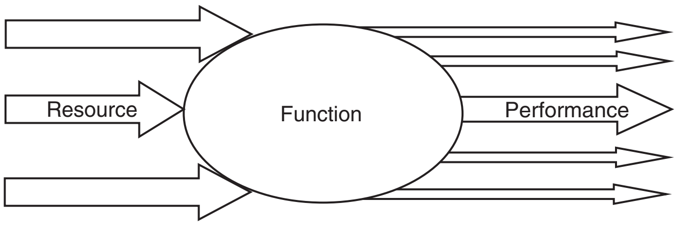


Figure 4.2
A simple system representation. It consists of a function, and its performance and resource (cost) attributes.

those performance attributes. The performance to cost ratio for a system is a measure of its efficiency.

Performance attributes are scalar. As discussed in Chapter 2, there are three basic types:

- **Quality**: The quality attributes specify *how well* the system performs. The term 'quality' is used here in the ordinary widest sense of the word. It is by no means limited to the narrow 'defect free' notion that some people mean when using it. How many qualities can you list of a great car, a dream house, an excellent employee, a great personal computer or a good wine? We include all such ideas in our broad concept of quality.
- **Resource Savings**: These specify *how much* resource is required to be saved compared to the resource usage/consumption by some reference or benchmark system. Achieving specific savings is frequently a driver for system development; for example, cutting the financial cost of carrying out transactions or reducing the time taken to carry out a task. Note, for this performance type, the *saving* of resource is the prime requirement, it is not simply a fortuitous by-product of the system improvements; it is what a stakeholder has specifically demanded.
- **Workload Capacity**: These specify *how much* work the system can perform: the capacity of the system. For example, the average speed for completing certain tasks, the capacity to store information and the maximum number of users supported.

Performance requirements must express *quantitatively* the stakeholders' requirements. I have come to believe, through experience, that all the performance attributes we want to control in real systems are capable of being expressed measurably. I find it intolerable that critical performance ideas are expressed in mere non-quantified words. Expressions like "vastly increased productivity" annoy me! Not one of those three words has a precise and agreed unambiguous interpretation. Yet, I have consistently encountered a world in multinational high-tech companies, amongst educated, intelligent and experienced people, where such vague expressions of performance, especially of quality, are tolerated; such expressions seem not even recognized as being dangerous and capable of improvement.

Performance attributes are more than a collection of names like 'reliability,' 'user friendliness,' 'innovation,' 'transaction time' and 'cost saving.' Each performance attribute needs to be precisely defined by a set of numeric, measurable, testable specifications. Each performance attribute specification will include different specified levels for different conditions [time, place and event]. Unless there is clear communication in terms of numeric requirements, there is every chance of the real requirements not being met; and we have no clear indication of the criteria for success and failure.

Sometimes, it seems difficult to identify satisfactory scales of measure. Often, the only ones that can be found are indirect, imprecise and have other problems associated with them. From my point of view, these problems can be tolerated. The specific scales of measure, and meters for measuring, can always be worked on and improved over time. In all cases, an attempt at quantified specification is better than vague words.

Over the years, I have found people immediately receptive to the idea that they should quantify all their performance ideas. The only question has been "How?" This chapter begins to answer this question. It describes the main Planguage parameters you can use to specify quantified performance attributes.

4.2 Practical Example: Performance Requirements

Let us start with an example of how to quantify a typical performance requirement.

"Increased ease of service" is a term I found in a set of design specifications for a mobile phone handset. It was not defined further. It sounded like a nice thing to have. The telecommunications supplier's culture allowed it to go unchallenged. In fact, in the few dozen pages of specification, there were actually 10 distinct design ideas, each one with a bullet-point claiming it would contribute to "better serviceability" for the new phone.

I asked my client if they had any quantified performance requirement specification for the "increased ease of service" for the phone. They had not, so we agreed to explore some possible definitions. We soon discovered that 'Serviceability' for the mobile phone had numerous elementary components; it was a *complex* objective.

Here is an extract of what we did. It was a three-step process.

Step 1

We identified the different components of Serviceability and gave each of them a name:

Serviceability: "is comprised of the following elementary and complex objectives."
Type: Quality Requirement:
    {Repair,
    Enhancement,
    Fashion Changes,
    Installation,
    Reconfiguration}.

Step 2

We described each of these objectives by defining and agreeing a Gist ('Gist' meaning the essence or main point):

Repair: Gist: Speed of repair of faults under given conditions.
Enhancement: Gist: Speed of introducing hardware or software enhancements.
Fashion Changes: Gist: Ease of customer varying fashion attachments.
Installation: Gist: Speed of installing telephone in defined settings (for example, in an automobile).
Reconfiguration: Gist: Work-hours to modify for varied uses (for example, coupling to personal computer or power supplies).

In fact, we then further decomposed these into a total of 18 elementary objectives. However, such detail is not required for this example!

Step 3

Once we were satisfied that we had captured the scope of Serviceability, we added further definition to specify the requirement in measurable and testable terms: we identified a Scale and a practical way of measuring on that Scale (a Meter). For example:

Repair:
Ambition: Improve the speed of repair of faults substantially, under given conditions.
Scale: Hours to repair or replace, from fault occurrence to when customer can use faultlessly, where they intended.
Meter [Product Acceptance]: A formal Field Test with at least 20 representative cases, [Field Audit]: Unannounced Field Test at random.
==================== Benchmarks ===================
Past [Product = Phone XYZ, Home Market, Qualified Dealer Shop]: {0.1 hours at Qualified Dealer Shop + 0.9 hours for the Customer to transit to/from Qualified Dealer Shop}.

Record [Competitor Product XX]: 0.5 hours average. "Because they drive a spare to the customer office."
Trend [USA Market, Large Corporate Users]: 0.3 hours. "As on-site spares for large customers."
====================== Targets =====================
Goal [Next New Product Release, Urban Areas, Personal Users]: 0.8 hours in total, [Next New Product Release, USA Market, Large Corporate Users]: 0.2 hours <- Marketing Requirement, February 3 This Year.
==================== Constraints ===================
Fail [Next New Product Release, Large Corporate Users]: 0.5 hours or worse on average <- Marketing Requirement, February 3 This Year.
Survival [Next New Product Release, All Markets]: 1.0 hours <- TG.

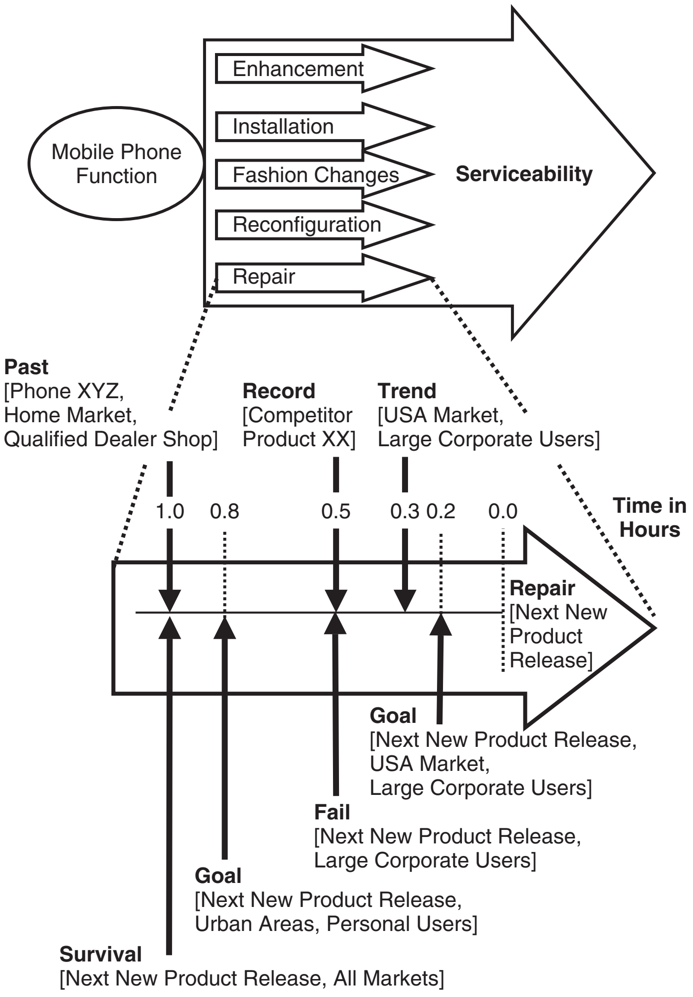

Mobile Phone Function
Enhancement
Installation
Fashion Changes
Reconfiguration
Repair
Serviceability
Past [Phone XYZ, Home Market, Qualified Dealer Shop]
Record [Competitor Product XX]
Trend [USA Market, Large Corporate Users]
1.0
0.8
0.5
0.3
0.2
0.0
Time in Hours
Repair [Next New Product Release]
Goal [Next New Product Release, USA Market, Large Corporate Users]
Fail [Next New Product Release, Large Corporate Users]
Goal [Next New Product Release, Urban Areas, Personal Users]
Survival [Next New Product Release, All Markets]

Figure 4.3
Serviceability, a complex performance requirement, decomposes into numerous performance attributes. One of these, the quality, Repair is expanded to show its targets and constraints and, the supporting benchmark information.

At this point, everyone realized that we needed to do some 'serious' work defining our Serviceability objective. Only with an improved requirement specification, could we begin to evaluate our ten specified design ideas that claimed "increased ease of service"!

Whoever had originally written the phrase "increased ease of service" had failed to communicate a precise, unambiguous requirement. Of considerable concern, there was clearly no means of agreeing the specific requirement level with other stakeholders. Nor was there any means of verifying we had met the requirement on delivery. In practice, the engineers were designing the phone without any significant input from Marketing.

These same problems, of 'lack of clarity' and 'lack of necessary detail', also occur elsewhere. In *your* business too!

4.3 Language Core: Scalar Attributes

All performance and resource/cost attributes are scalar. The Planguage parameters used for specifying scalar attributes[1] include:

- Ambition
- Scale
- Meter
- Past
- Record
- Trend
- Goal (abbreviation for 'Planned Goal' for performance attributes)
- Budget (abbreviation for 'Planned Budget' for resource attributes)
- Stretch (abbreviation for 'Stretch Goal' or 'Stretch Budget')
- Wish (abbreviation for 'Wish Goal' or 'Wish Budget')
- Fail
- Survival (abbreviation for 'Survival Limit').

Each scalar attribute must be specified using a tag, and an appropriate set of these parameters. Past, Record and Trend are used to specify *benchmarks*, Goal or Budget,[2] Stretch and Wish are used to specify *targets*, and Fail and Survival are used to specify *constraints*.

[1] For the sake of simplicity, only the abbreviations for these parameters tend to be used in the main text of this book. For example, where 'Stretch' is used, distinction is not made between 'Stretch Goal' and 'Stretch Budget' as it is evident from the context whether you are specifying a goal or a budget.

[2] In the past, 'Plan' was used instead of 'Goal' and 'Budget.' Use of 'Plan' is still perfectly acceptable if it better suits your organizational culture. 'Plan' does have the advantage of being simpler and of better conveying to people that it is a level that is subject to stakeholder acceptance and negotiation.

The numeric values of the target, constraint and benchmark parameters define the attribute levels. Note that benchmarks refer to *existing* or *past* values, or future estimates extrapolated from *past* values, whereas targets and constraints are *future requirement* values.

Here is some further detail about the main specification concepts:

Ambition

This is a descriptive parameter used to express the level of expected performance in words.

Scale

The heart of a scalar attribute specification is the 'scale of measure' parameter, Scale (sometimes also known as the 'units of measure' parameter). The Scale states the specific definition of a scalar attribute that we intend to use. It defines the units for measurement (for example, 'bits per second,' 'miles per hour,' 'mean time between failure' and 'number of new customer contracts per year').

While suitable scales of measure for resources/costs are relatively obvious to most people, identifying suitable Scales for performance attributes, especially for qualities, is more challenging. There are, as yet, few widely recognized standardized Scale definitions available. (*See further discussion in the next chapter, 'Scales of Measure'.*)

Scale: A scale of measure, stating the units to be used for numerically expressing a scalar attribute level.

Meter

The Scale parameter is supported by the Meter parameter, which defines, references or outlines a practical and economic method for actually carrying out the measurement of the numeric level of the attribute in a real system. More than one Meter may be required if the means of measuring is going to alter over time or vary according to the place and conditions.

Meter: A practical method for measuring and testing a scalar attribute level, on a defined Scale.

Benchmarks

The benchmark parameters, Past, Record and Trend are used for specifying historical, current or extrapolated performance levels. It is important that we understand what our own existing systems, our competitors' systems and, more generally, any other relevant systems, are achieving.

Past: A relevant benchmark level worth consideration that has already been achieved in some defined [time, place, event] conditions. We are interested in the levels achieved by any relevant existing system (our own, competitive, or any other system).

Record: A Past, which is the best-known result; a state-of-the-art numeric value.

Trend: An extrapolation of past data, trends and emerging technology to a defined [time, place, event] conditions. Aside from our own project's plans to improve this level, what future levels are likely to be achieved by others? What will we be competing with?

Targets

The target parameters, Goal or Budget, Stretch and Wish, define the attribute levels for success, motivation and dreams respectively.

Goal: A future required level under defined [time, place, event] conditions, which has to be achieved to claim success in meeting a performance attribute requirement. Goal levels are also a signal to stop investing in levels better than this one, as the value gained is most likely insufficient to justify additional costs.

Budget: A future required level under defined [time, place, event] conditions, which has to be achieved to claim success in staying within a resource attribute requirement. A signal to worry about resource usage when more resources are estimated to be needed, or are really used, than this 'acceptable' level of cost.

Stretch: A future desired level, under defined [time, place, event] conditions, which is designed to challenge people to exceed Goal levels or use less than Budget levels.

Wish: A future desired level, currently a 'dream' target level, under defined [time, place, event] conditions, which has some value to a stakeholder. The requirement is not planned or promised, due to technical or cost reasons, but it is recorded and kept in the requirement database (even if not acceptable now) so that it can be borne in mind.

Constraints

The constraint parameters, Fail and Survival, state the levels for some kind of system failure and system survival respectively.

Fail: A future required level under defined [time, place, event] conditions, that is necessary to avoid some kind and degree of system failure, while still allowing the system to operate.

Survival: A future required level under defined [time, place, event] conditions, which is necessary to avoid total system demise. In other words, it is a level necessary for the survival or viable operation of the system.

Conditions

It is also important to distinguish amongst the different numeric levels required for different conditions. Different times, different places and different events can all give rise to requirements for different attribute levels. Qualifiers should be used to specify the qualifying conditions for the different specified levels. Each of the above Planguage parameters will normally contain a qualifier and/or be within the scope of a qualifier defined elsewhere that applies to this specification. A qualifier can be used after almost any tag or parameter to be specific about dates, markets, products, components, stakeholders and other conditions. (*See also further discussion of qualifiers in Section 2.7.*)

[Time, Place, Event] or [When, Where, If]: Qualifiers specify the conditions for a specification being valid, in force or effective. All conditions must be 'true' for the associated specification to be valid.

Here is an example to illustrate the parameters just defined:

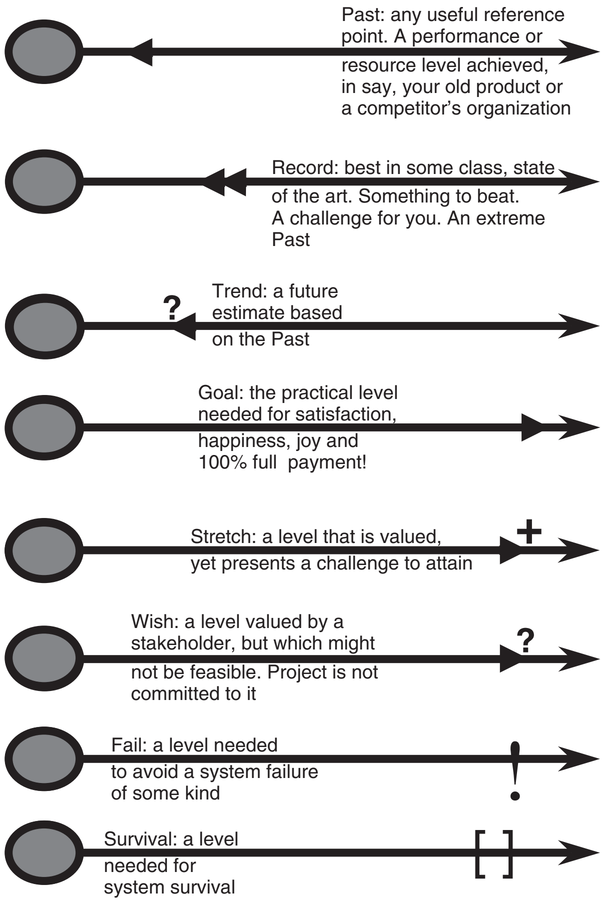
EXAMPLE

Usability [New Product Line, Major Markets]:
Ambition: To achieve a low average consumer time to learn to use our telephone answerer under stated conditions.
Scale: Average number of minutes for defined [Representative Consumers and all their household family members over 5 years old] to learn to use defined [Basic Daily Use Functions] correctly.
Meter [Product Acceptance]: A formal Field Test with at least 20 representative cases, [Field Audit]: Unannounced Field Test at random.
Past [Product XYZ, Home Market, People between 30 and 40 years old, in homes in Urban Areas, <for one explanation & demonstration>]: 10 minutes.
Record [Competitor Product XX, Field Trials]: <5 minutes?> <- one single case reported,
[USA Market, S Corporation]: 10 seconds <- Public Market Intelligence Report.
Fail [Next New Product Release, Children over 10]: 5 minutes <- Marketing Requirements February 3 Last Year.
Goal [Next New Product Release, Urban Areas, Personal Buyers]: 5 minutes, [Next New Product Release, USA Market, Large Corporate Users]: 5 minutes <- Marketing Requirements February 3 Last Year.
Stretch [Next Year]: (Record [USA Market] – 10%).

Table 4.1 Basic Planguage parameters for scalar attribute specification. See also Table 1.1 for the additional basic generic Planguage parameters.

Planguage Parameters for Scalar Attributes			
Scalar Attribute Parameter	*Meaning*	*Used for*	*Note also*
Scale:	Definition of the scale or units of measure	Defining the variable varying performance or cost concept with precision and clarity	Contractual use Icon: -l-l-
Meter:	Definition of how we are going to measure or test the level of this attribute	Determining the real world numeric levels in practice	Contractual use Icon: -l?l -
Past:	A known measured benchmark of an interesting past or current level	Providing a baseline attribute level	A useful reference point A benchmark Icon: <
Record:	A 'state of the art' level	If a Goal or Budget is near to or better than the Record, then a warning of extra risk and cost is implied	A useful reference point A benchmark Icon: <<
Trend:	From extrapolation of existing data, an estimated future level	A cross-check that the Goal or Budget level is ambitious/ competitive enough	A useful reference point A benchmark Icon: ?<
For performance attributes, Goal: *For resource attributes*, Budget:	A future, target requirement level, to be met or bettered for *success* and stakeholder *satisfaction*	Understanding the future requirement level. *Knowing when to stop* designing, investing and developing	A contractual *full* payment level A target Icon: >
Stretch:	An interesting, but difficult to attain, target level	To motivate teams to do better than currently considered practical or economic	No contractual commitment A target to strive towards as a challenge Icon: >+
Wish:	A desired level, dreamed of by some stakeholder, even if unrealistic	Better understanding of the stakeholder needs. Potential requirement direction when feasible later	No contractual commitment whatsoever – completely unbudgeted A stakeholder 'dream' to bear in mind Icon: >?
Fail:	A future requirement level, which is necessary to avoid *some sort* of system failure. A constraint	Stating a minimum requirement for delivery levels Understanding the minimum levels for any tradeoffs	Contractual use as a *minimum* payment level Icon: >>
Survival:*	A future requirement level, which is necessary to avoid *total* system failure. A constraint	Stating an absolute minimum requirement for any valid delivery or operation Failure to meet a Survival level means total system failure	Contractual use as a 'non-payment' level Icon: [ ] *Note this icon is deliberately similar to that used for qualifiers*

Note: * *The Catastrophe parameter is an alternative to the Survival parameter. See the Glossary. 'Survival' is used in the text of this book.*


Past: any useful reference point. A performance or resource level achieved, in say, your old product or a competitor's organization
Record: best in some class, state of the art. Something to beat. A challenge for you. An extreme Past
?
Trend: a future estimate based on the Past
Goal: the practical level needed for satisfaction, happiness, joy and 100% full payment!
Stretch: a level that is valued, yet presents a challenge to attain
+
Wish: a level valued by a stakeholder, but which might not be feasible. Project is not committed to it
?
Fail: a level needed to avoid a system failure of some kind
!
Survival: a level needed for system survival
[ ]

Note: This diagram applies to performance attributes and shows performance scale arrows. With a change to show scalar resource arrows, all the parameters also apply to resource attributes, apart from the Goal parameter, which is replaced by Budget for resource attributes.

Figure 4.4
Performance benchmarks, targets and constraints.

Note, terms defined by the project such as 'Major Markets' are capitalized to indicate that they are already, or will be shortly, more formally defined elsewhere. They will not necessarily be defined in these textbook examples. For example,

Major Markets: Defined As: {USA, Japan, Europe, India}.

There are other additional parameters that can be used to describe a scalar requirement. Some of these are shown in Figure 4.12, Scalar Requirement Template.

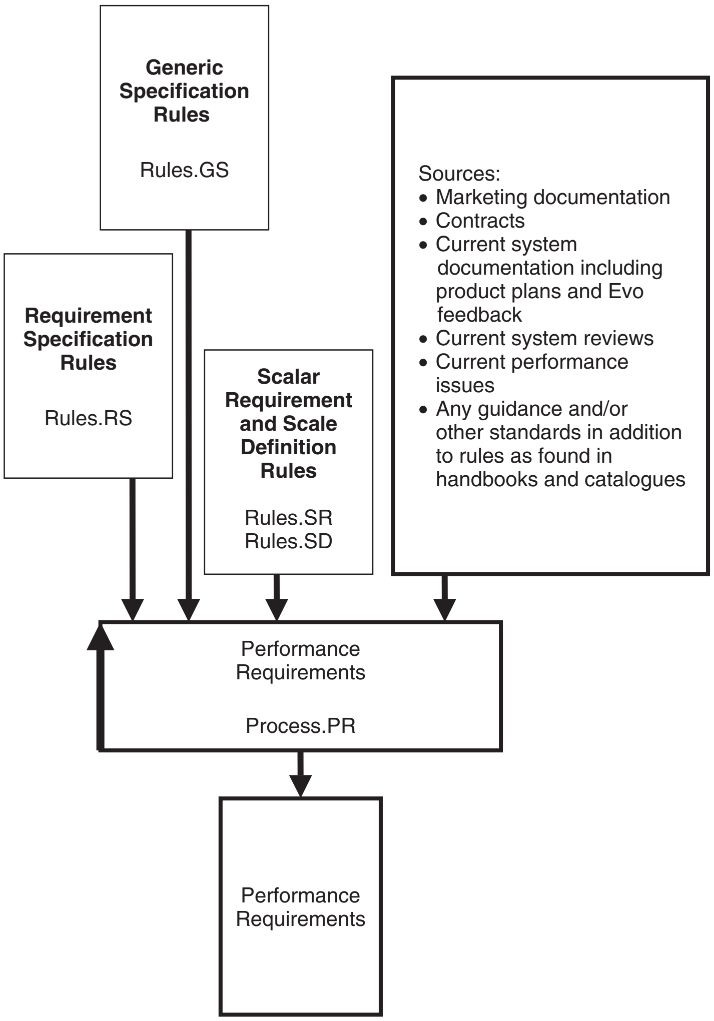

Generic Specification Rules
Rules.GS
Requirement Specification Rules
Rules.RS
Scalar Requirement and Scale Definition Rules
Rules.SR
Rules.SD
Sources:
• Marketing documentation
• Contracts
• Current system documentation including product plans and Evo feedback
• Current system reviews
• Current performance issues
• Any guidance and/or other standards in addition to rules as found in handbooks and catalogues
Performance Requirements
Process.PR
Performance Requirements

Figure 4.5
Performance requirements are subject to at least three types of rules for specification. These rules should be used in the SQC of performance requirements.

4.4 Rules: Scalar Requirements

Tag: Rules.SR.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: Rules for Scalar Requirement Specification.

Note: These rules apply to both performance requirement specification and to resource requirement specification.

Base: The generic rules for specification (Rules.GS), the rules for requirement specification (Rules.RS) and the specific rules for scale definition (Rules.SD) apply.

R1: **Completeness**: All scalar attributes, that are arguably critical to success or failure, shall be identified, specified and thoroughly defined.

R2: **Explode**: Where appropriate, a complex scalar requirement shall be specified in detail using a set of complex and/or elementary scalar attributes.

Note: In addition to detailing by means of elementary specifications, you can continue decomposing scalar specifications by using sets of [qualifiers].

R3: **Scale**: All elementary scalar attributes must define a single numeric Scale, fully and unambiguously, or reference such a definition.

R4: **Meter**: A *practical and economic* Meter, or set of Meters, shall be specified for tracking levels on each Scale. A reference to a full definition or standard measuring process for all identified Meters must be given. As an initial minimum for a new Meter, an outline of the Meter measuring process is permissible.

R5: **Benchmark**: A reasonable attempt shall be made to specify benchmarks {Past, Record, Trend} for our current system, and for relevant competitive systems. Explicit acknowledgement must be made where there is no known benchmark information.

R6: **Requirement**: At least one target level {Goal or Budget, Stretch, Wish} or Constraint {Fail, Survival} must be stated for a scalar attribute specification to classify as a *requirement* specification. *A specification with only benchmarks is an analytical specification, but not a requirement of any kind.*

R7: **Goal or Budget**: The numeric levels needed to meet requirements fully (and so achieve success) must be specified. In other words, one or more [qualifier defined] Goal or Budget targets must be specified. *The need for target levels to specifically cover all short term, long term and special cases must be considered.*

R8: **Stretch**: When you want to indicate an engineering challenge, in order to motivate design engineers to find designs to achieve better-than-expected levels, specify a 'Stretch' target (*using a Stretch parameter*). You should also include information about the benefits of reaching this target (*using Rationale*).

R9: **Wish**: Any known stakeholder wish level (a level that has some value to a stakeholder, but only a level to be dreamed of, it is an

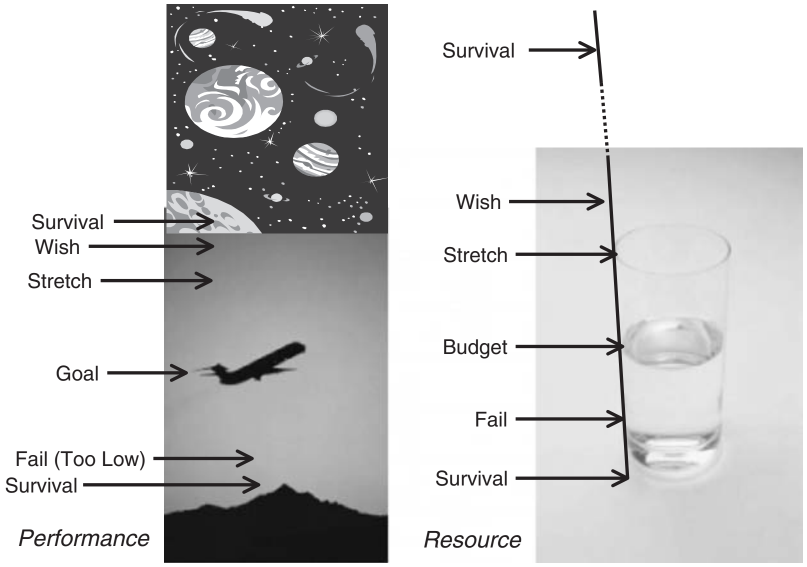


Figure 4.6
How the scalar requirement parameters can be used to describe real-world situations.

uncommitted level) shall be captured in a 'Wish' statement *(with Rationale). Even if the Wish level cannot realistically yet be converted into a practical target level, it is valuable competitive marketing information and may allow us to better satisfy the stakeholder at some future point.*

R10: **Fail**: Any known numeric levels to *avoid system, political, legal, social, or economic loss or pain* must be specified. In other words, one or more [qualifier defined] Fail constraints must be specified. *Several Fail levels may be useful for a variety of short term, long term, and special situations.*

R11: **Survival**: The numeric levels to avoid complete system failure (a totally unusable or unrecoverable system) must be specified. In other words, any [qualifier defined] constraint levels at which system survival is completely at risk must be identified, using Survival parameters.

4.5 Process Description: Performance Requirements

Process: Performance Requirements.

Tag: Process.PR.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Entry Conditions

E1: The Generic Entry Conditions apply. The list of valid source documents could include marketing documentation, contracts, current system documentation, current system reviews, any lists of system performance issues and any initial design specifications. It specifically includes any documentation of standards that applies, such as handbooks and catalogues. The required specification standards include {Rules.GS, Rules.RS, Rules.SR and Rules.SD}.

Procedure

P1: Scan all input (source) documents for implied (for example, via design specifications) or *explicitly expressed* performance requirements. Build a list of performance requirements categorized by stakeholder type.

P2: Next, scan all input (source) documents (including any design documents and strategic plans) for design ideas. Mark the design ideas as requirements, ONLY if they are intentional design constraints (*as they are then true requirements*). Otherwise, if they are not constraints, determine and specify the possible performance requirements that led to these design ideas being specified. Add these 'implied' performance requirements to the overall list of requirements. *You can keep the design ideas, separately, for design phases. But get them out of the real requirements. You might well cross-reference the implied requirements (Impacts: <Requirement X>.) and design suggestions (Is Supported By: <Design Idea Y>.) for future understanding of why they are there.*

P3: Using the P1 lists (*explicit requirements*) and P2 lists (*requirements derived from design ideas*), establish a comprehensive list of candidate performance requirements. Specify at least Tag, and possibly a Gist or Ambition. Include cross-reference to any Sources (<-), Assumptions, Dependencies and Risks you can determine.

P4: Check handbooks, catalogues and lists of standard performance requirements for ideas of additional performance requirements, which you should consider. *Remember that you need to specify things that are currently taken for granted because they are not problems in any of your current products or systems. We have to keep our system healthy in the future, consciously!*

P5: Consider the total stakeholder environment. This involves not just your one or two users and customers, but more likely, your ten or more internal and external stakeholders, such as help desk, installers, politicians, marketing and customer trainers. Using the input documents, brainstorm to determine each stakeholder's critical qualitative attributes. Ensure that the critical performance attributes are identified on your requirements' list. Interview the stakeholders to get feedback and confirmation about your specification. Add to the list of requirements or modify them as necessary.

P6: For each identified performance requirement, specify it in detail using the rules that apply (Rules.GS, Rules.RS, Rules.SR and Rules.SD). Ensure each performance attribute is measurable in practice.

P7: Consider which performance requirements are key, and must therefore be controlled. Identify the most important 'top ten' performance requirements. Group the others as 'Diverse' or 'Less Critical' if you like.

P8: Perform Specification Quality Control (SQC) on the performance requirements. Correct any identified defects, and calculate the remaining major defects/page (a page being 300 words of non-commentary text). Check against the rules: {Rules.GS, Rules.RS, Rules.SR and Rules.SD}.

Exit Conditions

X1: The Generic Exit Conditions apply. The maximum possibly remaining major defects/page must be less than one.

X2: The Requirement Specification Owner (usually specified as 'Owner: <name, e-mail address or department>.') agrees to release the performance requirement specification with *their* name on it. They have veto on release.

4.6 Principles: Performance Requirements

1. **The Principle of 'Bad numbers beat good words'**
   Poor quantification is more useful than no quantification; at least it can be improved systematically.
2. **The Principle of 'Performance quantification'**
   All performance attributes can be expressed quantitatively, '*qualitative*' does not mean unquantifiable.
3. **The Principle of 'Threats are measurable'**
   If the lack of a performance attribute can destroy your project, then you can measure it sometime; the only issue will be "how early?"
4. **The Principle of 'Put up or shut up'**
   There is no point in demanding a performance requirement, if you cannot pay or wait for it.
5. **The Principle of 'Deadline or die'**
   There is no point in demanding a performance requirement, if you would always give priority to something else, for example, a deadline.

6. **The Principle of 'Dream, but don't hold your breath'**
   There is no point in demanding a performance requirement, if it is outside the state of your art.

7. **The Principle of 'Benchmarks and targets'**
   Numeric past 'history' levels and numeric future requirement levels *together complete* the performance requirement definition of relative terms like 'improved'.

8. **The Principle of 'Scalar priority'**
   In practice, the first priority will be survival,
   The second priority will be avoiding failure,
   The third priority will be success,
   And the required levels for all of these will be constantly changing.

9. **The Principle of 'Many-splendored things'**
   Most performance ideas are usefully described by *several* measures of goodness.

10. **The Principle of 'Limits to detail'**
    There is a *practical* limit to the number of facets of performance you can define and control.
    It is far less than the number of facets that you can *imagine* might be relevant. (Try a limit of just the Top Ten!)

4.7 Additional Ideas: Performance Requirements

Handling Complex Performance Requirements

Many performance requirements, like the quality requirement, 'Usability', can be expressed in greater detail using sub-requirements (such as Learning Time, Error Rate and Minimum Skills Entry Level). There are many possible interpretations, and they all have some use or validity. We call such decomposable ideas 'complex requirements'. It would be easy to think, "there is no measure to cover such a complex requirement." Our attitude is pragmatic and says, "We *will* define a reasonable number of the sub-requirements quantitatively, and *use* them to define what we mean." We only need to identify sufficient sub-requirements to capture the meaning of the performance attribute in the current system context.

The Planguage structure for a hierarchy is as follows:

Tag:

Sub-Tag 1: Scale: <some scale>. <Other parameters as needed>,

Sub-Tag 2: Scale: <some other scale>. <Other parameters as needed>,...

Sub-Tag n: Scale: <yet another scale definition>. <Other parameters as needed>.

For example, the first step of the practical example given in Section 4.2 is primarily discussing this idea of expanding a complex quality requirement, 'Serviceability', into a number of elementary and complex quality requirements ('Repair', 'Enhancement', 'Fashion Changes', 'Installation' and 'Reconfiguration').

Here are some *known* 'classic' decomposition examples in the form of a {descriptive tag set}:

EXAMPLE Performance: {Quality: {Availability {Reliability, Maintainability, Integrity}, Adaptability, Usability}, Resource Saving, Work Capacity: Storage Capacity}.

EXAMPLE Maintainability: {Problem Recognition, Administrative Delay, Tool Collection, Problem Analysis, Change Specification, Quality Control, Modification Implementation, Modification Testing, Recovery}.

EXAMPLE Usability: {Entry Level Experience, Training Requirements, Handling Ability, Likeability, Demonstrability}.

(See also the next chapter, especially Section 5.7.)

Limit the Amount of Detail

Expanding complex performance requirements into a number of sub-requirements (and the subsequent need to further expand any sub-requirements that are also themselves complex requirements) usually leads to a great deal of detailed information when specification of the parameters is carried out.

Make sure you focus on the critical (key-influence) performance attributes. Tracking the top ten attributes is usually more than sufficient to make a start. Remember, if you are using evolutionary delivery, you can always decide to modify which attributes you are monitoring over time.

Setting Scalar Levels

Implicit Assumptions Supporting a Scalar Parameter Level

When you set a scalar level, there are certain *implicit* supporting assumptions, which apply. For example, when you specify a Goal level, you are very unlikely to mean 'this level and only this level.' You actually are specifying that a stakeholder wants 'this level *or better*.' Of course, all the other simultaneously specified targets and

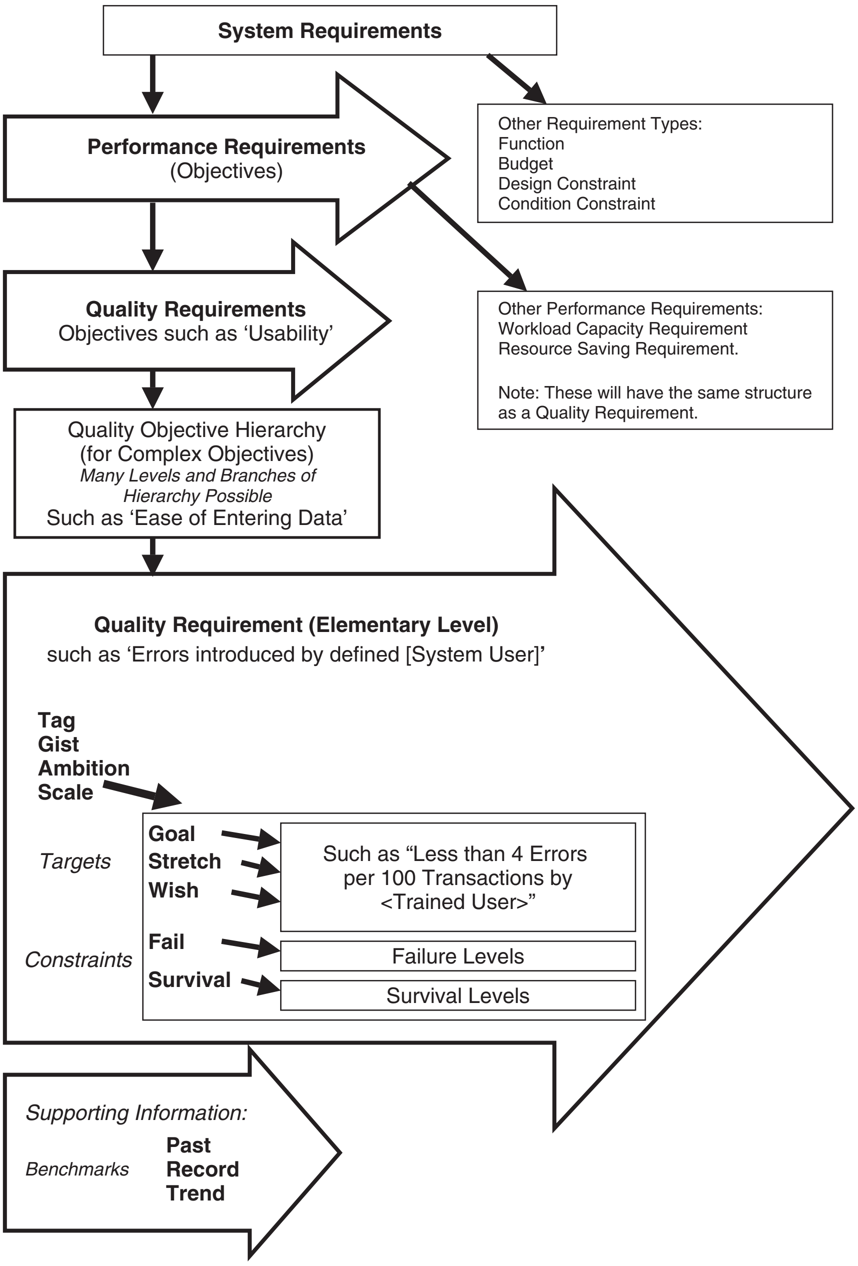

System Requirements
Performance Requirements (Objectives)
Other Requirement Types: Function Budget Design Constraint Condition Constraint
Quality Requirements Objectives such as 'Usability'
Other Performance Requirements: Workload Capacity Requirement Resource Saving Requirement.
Note: These will have the same structure as a Quality Requirement.
Quality Objective Hierarchy (for Complex Objectives)
Many Levels and Branches of Hierarchy Possible
Such as 'Ease of Entering Data'
Quality Requirement (Elementary Level)
such as 'Errors introduced by defined [System User]'
Tag
Gist
Ambition
Scale
Targets
Goal
Stretch
Wish
Such as "Less than 4 Errors per 100 Transactions by <Trained User>"
Constraints
Fail
Failure Levels
Survival
Survival Levels
Supporting Information:
Benchmarks
Past
Record
Trend

Figure 4.7
Requirement specification hierarchy for a quality requirement.

Table 4.2 A teaching example supplied by Erik Simmons, Intel. The data is not real! Note the explicit direction specified for the Fail levels.

Attribute	*Parameter*		
	Fail	*Goal*	*Stretch*
Performance			
Power (watts)	>10 W	5 W	3–4 W
Product Cost (each unit)	>$21.85	$21.60	$21.50
MTTF (hrs)	<10,000	20,000	25,000
Battery (hrs)	<8	12	16
Weight (lbs)	>5	3	2
Display (diagonal in inches)	<7	8	9
Resource			
Ship Date	>March Next Year	January Next Year	November This Year
Effort (hrs)	>25,000	23,000	22,000
Peak Headcount	>15	12	10

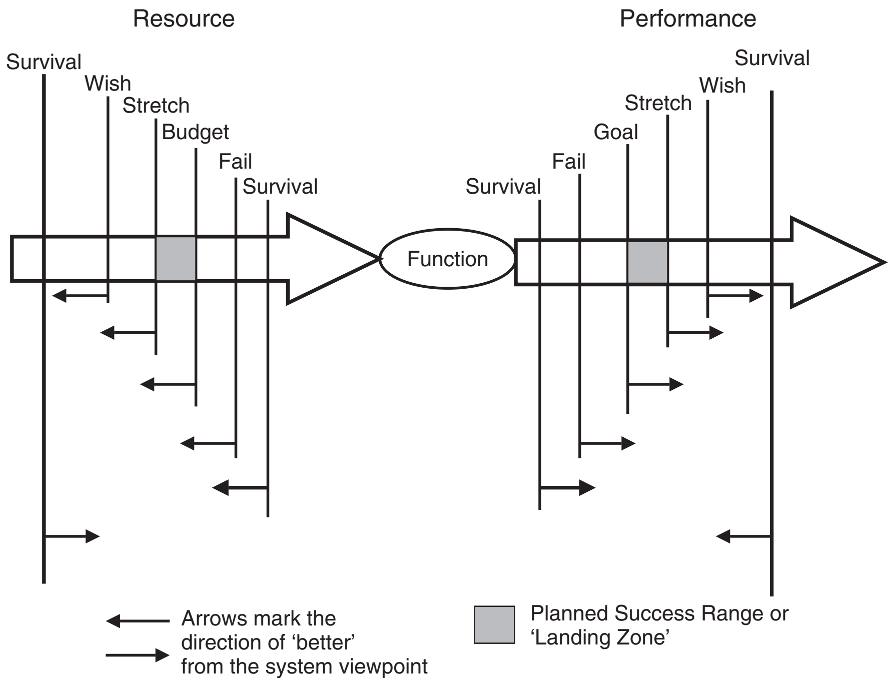

Resource
Performance
Survival
Wish
Stretch
Budget
Fail
Survival
Function
Survival
Fail
Goal
Stretch
Wish
Survival
Arrows mark the direction of 'better' from the system viewpoint
Planned Success Range or 'Landing Zone'

Figure 4.8
Implicit direction for 'better' along a Scale.

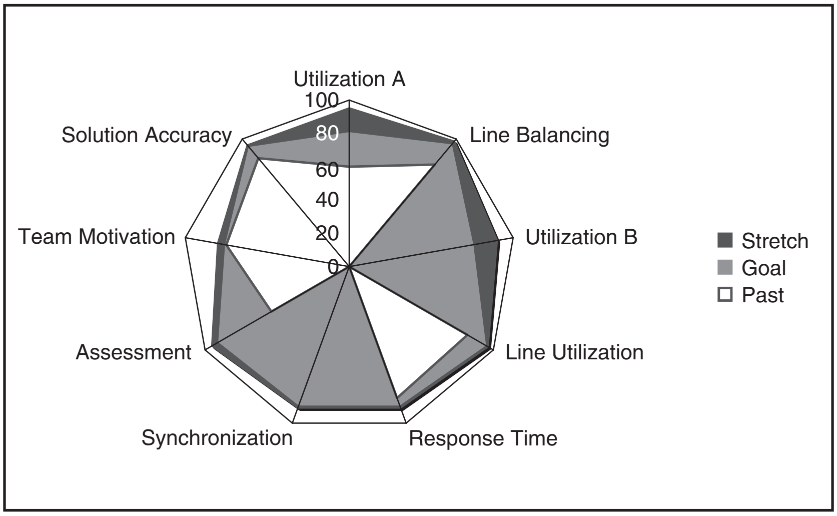


Figure 4.9
A real case study diagram (slightly modified to preserve anonymity) showing a net of multidimensional performance scales of measure. It shows a snapshot of a system at a specific time. The areas show the Past [At Current Time], the gap from Past to the Goal targets, and the gap from the Goal targets to the Stretch targets. This is a powerful graphical way of displaying scalar data.
Note: Resources are not shown and the Performance scalar arrows are spread through 360 degrees.

constraints that apply under the same set of conditions have to be taken into account as well: the stakeholder wants *all* these requirements at the same time. By specifying the Goal level, the stakeholder is providing the information about what they consider the *minimum* performance level for *success* in the light of the other requirements.

Exactly what 'or better' means in numeric terms depends on your Scale definition. A stakeholder wants *more* performance and to use *less* resource (see Figure 4.8). However, the Scale finally dictates the 'direction' of the numeric value and, therefore, the numeric interpretation of 'better.' For example, 'better' performance can mean a reduction in the time taken to carry out a task – a numeric level would therefore be expected to *reduce* over time as performance improved along the Scale.

4.8 Further Example/Case Study: Performance Specification for a Water Supply

Here is a real example of specifying Norwegian Church Aid's performance requirements (objectives) for improving the water supply in Eritrea.

Function: Supplying Water [Eritrea] <- Norwegian Church Aid (NCA).

We began by capturing the immediate objectives:

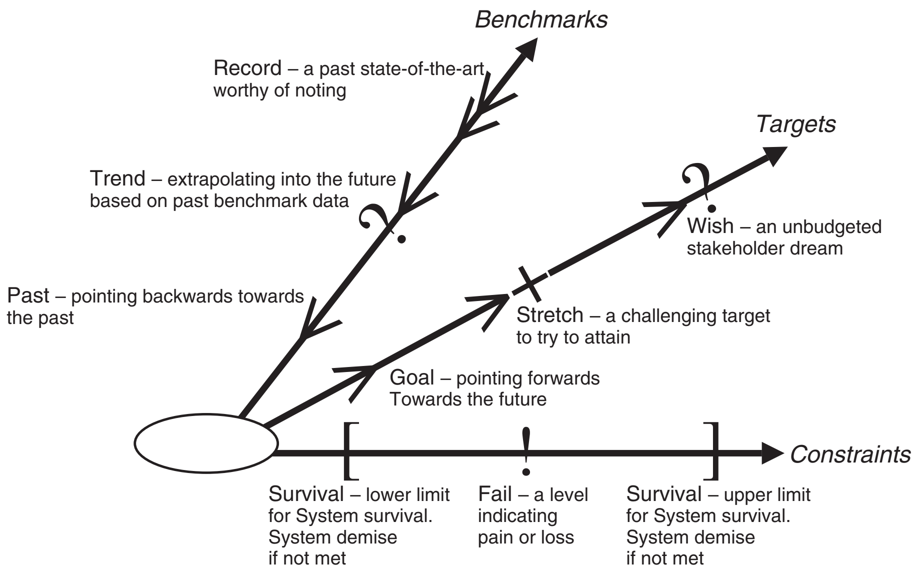
EXAMPLE **Operation and Maintenance**
Local Control:
Ambition: Strengthen conditions for local management of Operation and Maintenance.
Scale: % of Water Supply Pumps which <function> more than 23 hours out of each 24-hour period.
Meter: A <status report> from the Local Water Committees every quarter year.
Past [Eritrea, Four Years Ago]: 65 ± 5% <- Survey conducted by NCA's health co-ordinator.
Goal [Eritrea, By End of this Year]: 80%,
[Eritrea, By End of Next Year]: 90% <- NCA Planning Committee [May Last Year].
Pump Availability:
Ambition: No single Water Supply Pump shall be <out of order> for <a long period of time>.
Scale: % of year Water Supply Pumps <function>.
Meter: Faults reported by the Local Water Committees and the Water Supply Projects.
Past [Eritrea, Four Years Ago]: 60 ± 40% <- ?
Goal [Eritrea, By End of This Year]: 90 ± 10% <- ?,
[Eritrea, By End of Next Year]: 95 ± 5%.

Water Supply Efforts
Well Rehabilitation:
Ambition: Rehabilitation of earlier water supply projects and efforts.
Scale: Number of Water Supply Pumps put into operation anew each year, which satisfy the <minimum need>.
Meter: Reports by Local Water Committees every quarter year.
Past [Eritrea, Four Years Ago]: 30 ± 5%. "of a total of 300."
Goal [Eritrea, By End of This Year]: 40 ± 5% <- ?,
[Eritrea, By End of Next Year]: 35 ± 5%.
New Wells:
Ambition: Make newly drilled wells when other alternatives are not feasible.
Assumptions: {1. New Wells are only to be drilled when other alternatives are impossible. 2. Institutional responsibility and participation from the local village shall be defined and accepted in advance.} <- NCA Policy.
Scale: Number of New Wells completed by agreed dates and according to the Contract between the Drilling Team and the Employer.
Meter: Reports by Local Water Committees every quarter year.
Past [Eritrea, Seven Years Ago]: 66,
[Eritrea, Six Years Ago]: 17.
Goal [Eritrea, By End of This Year]: 10,
[Eritrea, By End of Next Year]: 9.
Alternative Sources:
Gist: Alternatives to drilled wells will be developed whenever the situation permits it.
Scale: Number of efforts per year, which result in <alternative water supplies>.
Meter: Reports by Local Water Committees, Aid Partners or Aid Projects every quarter year.
Past [Eritrea, Five Years Ago]: 20,
[Eritrea, Four Years Ago]: 19.
Goal [Eritrea, By End of This Year]: 30,
[Eritrea, By End of Four Years]: 46.

Once we had captured these objectives, we were pleased that we had a clear statement of the requirements that could easily be used for planning purposes and could readily be monitored. However, we soon realized that these goals were *not* directly specifying people's needs; for example, improvement in health, clean water and ease of getting the water to where it should go. Suggestions were consequently made for improved goal setting with a series of new scales. For example, 'average time to pick up the water' and '% of people that die/get sick due to unclean water.'

The major result of the specification was the recognition that the high-level aims of the water projects needed better definition, and that the water projects needed to be seen in that light.

4.9 Diagrams/Icons: Scalar Attribute Requirements


Benchmarks
Record – a past state-of-the-art worthy of noting
Trend – extrapolating into the future based on past benchmark data
Past – pointing backwards towards the past
Targets
Wish – an unbudgeted stakeholder dream
Stretch – a challenging target to try to attain
Goal – pointing forwards Towards the future
Constraints
Survival – lower limit for System survival. System demise if not met
Fail – a level indicating pain or loss
Survival – upper limit for System survival. System demise if not met

Note: A Scale icon is drawn as a line with an arrowhead, connected to a function oval symbol. Performance scales are to the right from the function oval (O→), and resource scales are at the left of the oval with arrowhead connected to the oval (→O). The performance and resource attribute icons must both include a function icon (an oval) to distinguish them from each other. The arrow in a performance attribute points away from the function oval. For a resource attribute, the arrow points towards the function oval.

Figure 4.10
Three graphical performance attributes showing the icons for scalar performance attribute levels: three analytical benchmarks, three future requirement targets and two future requirement constraints, respectively. Usually an attribute would have a mix of whatever benchmark, target and constraint levels were relevant.

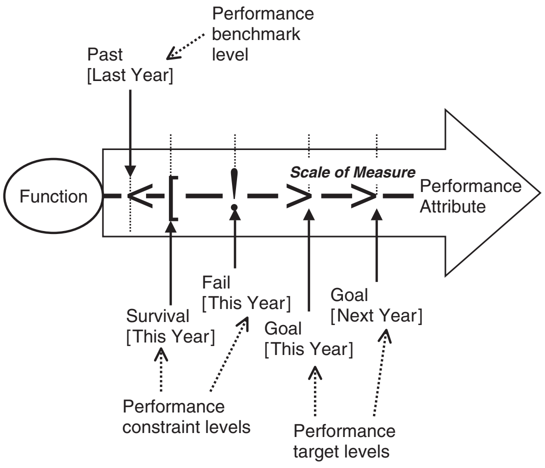


Figure 4.11
Example of using some of the scalar icons: two performance target levels and two constraint levels compared to one benchmark level.

Table 4.3 Icons for scalar attribute requirements.

Planguage Term Attribute Definition	*Icon*
Gist	Σ
Ambition	@.Σ
Scale	-|-|-
Meter	-|?|-
Targets	
Goal or Budget	>
Stretch	>+
Wish	>?
Constraints	
Fail	!
Survival	[ ]
System Space Conditions	
Time, Place and Event	[qualifier conditions]
Supporting Information	
Source	<-
Comment	"text."
Benchmarks	
Past	<
Record	<<
Trend	?<

Elementary scalar requirement template <with hints>

Tag: <Tag name of the elementary scalar requirement>.

Type:
<{Performance Requirement: {Quality Requirement,
Resource Saving Requirement,
Workload Capacity Requirement},
Resource Requirement: {Financial Requirement,
Time Requirement,
Headcount Requirement,
others}}>.

============================ Basic Information ============================
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Role/e-mail/name of the person responsible for this specification>.

Stakeholders: <Name any stakeholders with an interest in this specification>.

Gist: <Brief description, capturing the essential meaning of the requirement>.
Description: <Optional, full description of the requirement>.
Ambition: <Summarize the ambition level of *only the targets* below. Give the overall real ambition level in 5–20 words>.

============================ Scale of Measure ============================
Scale: <Scale of measure for the requirement (States the units of measure for all the targets, constraints and benchmarks) and the scale qualifiers>.

============================= Measurement =============================
Meter: <The method to be used to obtain measurements on the defined Scale>.

============= Benchmarks ============= "Past Numeric Values" =============
Past [<when, where, if>]: <Past or current level. State if it is an estimate> <- <Source>.
Record [<when, where, if>]: <State-of-the-art level> <- <Source>.
Trend [<when, where, if>]: <Prediction of rate of change or future state-of-the-art level> <- <Source>.

============== Targets ============== "Future Numeric Values" =============
Goal/Budget [<when, where, if>]: <Planned target level> <- <Source>.
Stretch [<when, where, if>]: <Motivating ambition level> <- <Source>.
Wish [<when, where, if>]: <Dream level (unbudgeted)> <- <Source>.

============== Constraints ============= "Specific Restrictions" =============
Fail [<when, where, if>]: <Failure level> <- <Source>.
Survival [<when, where, if>]: <Survival level> <- <Source>.

============================= Relationships =============================
Is Part Of: <Refer to the tags of any supra-requirements (complex requirements) that this requirement is part of. A hierarchy of tags (For example, A.B.C) is preferable>.
Is Impacted By: <Refer to the tags of any design ideas that impact this requirement> <- <Source>.
Impacts: <Name any requirements or designs or plans that are impacted significantly by this>.

====================== Priority and Risk Management ======================
Rationale: <Justify why this requirement exists>.
Value: <Name [stakeholder, time, place, event]: Quantify, or express in words, the value claimed as a result of delivering the requirement>.
Assumptions: <State any assumptions made in connection with this requirement> <- <Source>.
Dependencies: <State anything that achieving the planned requirement level is dependent on> <- <Source>.
Risks: <List or refer to tags of anything that could cause delay or negative impact> <- <Source>.
Priority: <List the tags of any system elements that must be implemented before or after this requirement>.
Issues: <State any known issues>.

Figure 4.12
A scalar requirement template with hints.

4.10 Summary: Performance Requirements

The basic initial step to get control over the primary ‘drivers’ for plans and resulting projects is to have a clear specification of what we want.

Consider:

- Performance requirements are often ‘hidden’ in undefined requirement terms, such as ‘increased adaptability’.
- Performance requirements may be hidden in designs and plans that have been inadvertently specified amongst the requirements. For example ‘Flexible Contracts’ is a design idea seeming to imply that there is some (undefined) form of ‘flexibility’ required, but what is it?
- Performance requirements need to be *numeric* and to be *qualified by conditions*, so we can specify *exactly* what stakeholders want and the [time, place and event] conditions that we must meet.
- Performance requirements must be specified in such a way that they are testable.
- Performance levels are variable; they change from project to project and vary within a project over time, place and events.

Performance requirements are the key statements of expected and necessary critical stakeholder benefits for a project. Performance requirements are the main reason why projects are funded at all. So it is critical that they are done well and managed well.

Chapter
5

SCALES OF MEASURE
How to Quantify

GLOSSARY CONCEPTS
Scale
Meter

5.1 Introduction

Scales of measure are fundamental to Planguage. They are central to the definition of all scalar attributes, that is, to all the performance and resource attributes.

You should learn the art of developing your own *tailored* scales of measure for the performance and resource attributes, which are important to your organization or system. You cannot rely on being 'given the answer' about how to quantify. You would soon lose control over your current vital concerns if you waited for that!

Finding and Developing Scales of Measure and Meters

The basic advice for identifying and developing scales of measure and meters (practical methods for measuring) for scalar attributes is as follows:

1. Try to 'reuse' previously defined Scales and Meters. *See Figure 5.3, Examples of Scales of Measure.*
2. Try to 'modify' previously defined Scales and Meters.
3. If no existing Scale or Meter can be reused or modified, use common sense to develop innovative home-grown quantification ideas.
4. Whatever Scale or Meter you start off with, you must be prepared to learn. Obtain and use early feedback, from colleagues and from field tests, to redefine and improve your Scales and Meters.

See also Section 5.5, 'Process Description: Scale Definition.'

Reference Library for Scales of Measure

'Reuse' is an important concept for sharing experience and saving time when developing Scales. You need to build reference libraries of your 'standard' scales of measure. Remember to maintain details supporting each standard Scale, such as Source, Owner, Status and Version (Date). If the name of a Scale's designer is also kept, you can probably contact them for assistance and ideas.

EXAMPLE

Tag: <Assign a tag to this Scale>.
Type: Scale.
Version: <Date of the latest version or change>.
Owner: <Role/email of person responsible for updates/changes>.
Status: <Draft, SQC Exited, Approved>.
Scale: <Specify the Scale with defined [qualifiers]>.
Alternative Scales: <Reference by tag or define other Scales of interest as alternatives and supplements>.

Embedded Scale Qualifiers: <Define the scale parameters, list options>.
Meter Options: <Suggest Meter(s) appropriate to the Scale>.
Known Usage: <Reference projects & specifications where this Scale was actually used in practice with designers' names>.
Known Problems: <List known or perceived problems with this Scale>.
Limitations: <List known or perceived limitations with this Scale>.
This is a draft template with hints for specification of scales of measure in a reference library.

Reference Library for Meters

Another important standards library to maintain is a library of 'Meters.' Meters (*as discussed in Chapter 4*) support scales of measure by providing practical methods for actually measuring the numeric Scale values. 'Off the shelf' Meters from standards' reference libraries can save considerable amounts of time and effort; they are already developed and are 'tried and tested' in the field.

It is natural to reference suggested Meters within definitions of specific scales of measure (as in the template above). Scales and Meters belong intimately together.

EXAMPLE Tag: Ease of Access.
Type: Scale.
Version: <*version date*>.
Owner: Rating Model Project (Bill).
Scale: Speed for a defined [Employee Type] with defined [Experience] to get a defined [Client Type] operating successfully from the moment of a decision to use the application.
Alternative Scales: None known yet.
Embedded Scale Qualifiers:
  Employee Type: {Credit Analyst, Investment Banker, . . . }.
  Experience: {Never, Occasional, Frequent, Recent}.
  Client Type: {Major, Frequent, Minor, Infrequent}.
Meter Options:
  Test all frequent combinations of parameters at least twice. Measure speed for the combinations.
Known Usage: Rating Model Project.
Known Problems: None recorded yet.
Limitations: None recorded yet.
Example of a 'Scale' specification for a reference library.

Managing 'What' You Measure

It is a well-known paradigm that you can manage what you can measure. If you want to achieve something in practice, then quantification, and later measurement, are essential first steps for making sure you get it. If you do not make it measurable, then it is likely to be less motivating for people to find ways to deliver it (they have no clear targets to work towards and there are not such precise criteria for judgment of failure or success).

On Quantification

- No matter how complex the situation, good systems engineering involves putting value measurements on the important parameters of desired goals and performance of pertinent data, and of the specifications of the people and equipment and other components of the system.
- It is not easy to do this and so, very often, we are inclined to assume that it is not possible to do it to advantage.
- But skilled systems engineers can change evaluations and comparisons of alternative approaches from purely speculative to highly meaningful.
- If some critical aspect is not known, the systems experts seek to make it known. They go dig up the facts.
- If doing so is very tough, such as setting down the public's degree of acceptance among various candidate solutions, then perhaps the public can be polled.
- If that is not practical for the specific issue, then at least an attempt can be made to judge the impact of being wrong in assuming the public preference.
- Everything that is clear is used with clarity: what is not clear is used with clarity as to the estimates and assumptions made, with the possible negative consequences of the assumptions weighed and integrated.
- We do not have to work in the dark, now that we have professional systems analysis.

Simon Ramo

Figure 5.1
A quote by Simon Ramo of TRW (Ramo and St. Clair 1998 Page 81).

5.2 Practical Example: Scale Definition

'User-friendly' is a popular term. Can you specify a scale of measure for it?

Here is my advice on how to tackle developing a definition for this.

If we assume there is no 'off-the-shelf' definition that could be used:

1. Be more specific about the various aspects of the quality 'user-friendly' that are to be tackled. There are many, but decide on about 5 to 15 in practice that are key to your environment. For this example, let's select 'environmentally friendly' as the one of many aspects that we are interested in, and we shall work on this below as an example. (There are many other elementary aspects of the complex requirement, 'User Friendly', which we could also have chosen.)
2. Invent and specify a Tag: 'Environmentally Friendly' is sufficiently descriptive. Ideally, it could be shorter, but it is very descriptive left as it is.

EXAMPLE Tag: Environmentally Friendly.

Note, we usually don't explicitly specify 'Tag:'.

3. Check there is an Ambition statement, which briefly describes the level of requirement ambition.

EXAMPLE Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things.

4. Ensure there is general agreement by all the involved parties with the Ambition. If not, ask for suggestions for modifications or additions to it. Here is a simple improvement to my initial Ambition statement. It actually introduces a 'constraint'.

EXAMPLE Ambition: A high degree of protection, compared to competitors, over the short term and the long term, in near and remote environments for health and safety of living things, **which does not reduce the protection already present in nature**.

5. Using the Ambition description, define an initial Scale that is somehow measurable. Think about what will be perceived by the stakeholders if the level of quality changes. What would be a visible effect if the quality improved? My initial unfinished attempt at finding a suitable Scale captured the ideas of change occurring and of things getting better or worse:

EXAMPLE Scale: The % change in positive (good environment) or negative directions for defined . . .

However, I was not happy with it, so I made a second attempt. I refined the Scale by expanding it to include the ideas of specific things being effected in specific places over given times:

EXAMPLE Scale: % destruction or reduction of defined [Thing] in defined [Place] during a defined [Time Period].

This felt better. In practice, I have added [qualifiers] into the Scale, to indicate the variables that must be defined by specific things, places and time periods whenever the Scale is used.

6. Determine if the term needs to be defined with several scales of measure, or whether one like this, with general parameters, will do. Has the Ambition been adequately captured? To determine what's best, you should list some of the possible sub-components of the term (that is, what can it be broken down into, in detail?). For example:

EXAMPLE Thing: {Air, Water, Plant, Animal}.
Place: {Personal, Home, Community, Planet}.
Alternatively,
Thing: = {Air, Water, Plant, Animal}.
Place: Consists of {Personal, Home, Community, Planet}.

The first example means: 'Thing' is defined as the set of things Air, Water, Plant and Animal (which since they are capitalized are themselves defined elsewhere). Instead of just the colon after the tag, the more explicit Planguage parameter 'Consists Of' or '=' can be used to make this notation more immediately intelligible to novices in reading Planguage.

Then consider whether the Scale enables the performance levels for these sub-components to be expressed. You may have overlooked an opportunity and may want to add one or more qualifiers to that Scale. For example, we could potentially add the scale qualifier '. . . under defined [Environmental Conditions] in defined [Countries] . . . ' to make the scale definition even more explicit and more general.

Scale qualifiers (like . . . 'defined [Place]' . . .) have the following advantages:

- they add clarity to the specifications
- they make the Scales themselves more reusable in other projects
- they make the Scale more useful in this project: specific benchmarks, constraints and targets can be made for any interesting combination of scale variables (such as, 'Thing = Air').

7. Start working on a Meter (remember, you should first check there is not a standard or company reference library Meter that you could use). Try to imagine a practical way to measure things along the Scale, or at least sketch one out. My example is only an initial rough sketch.

EXAMPLE Meter: {scientific data where available, opinion surveys, admitted intuitive guesses}.

The Meter will help confirm your choice of Scale as it will provide evidence that practical measurements can feasibly be obtained using the Scale.

8. Now try out the Scale. Define some reference points from the past (*benchmarks*) and some future requirements (*targets* and *constraints*).

EXAMPLE Environmentally Friendly:
Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things, which does not reduce the protection already present in nature.
Scale: % destruction or reduction of defined [Thing] in defined [Place] during a defined [Time Period].
======================== Benchmarks ========================
Past [Time Period = Next Two Years, Place = Local House, Thing = Water]: 20% <- intuitive guess.
Record [Last Year, Cabin Well, Thing = Water]: 0% <- declared reference point.
Trend [Ten to Twenty Years From Now, Local, Thing = Water]: 30% <- intuitive. "Things seem to be getting worse."

========================== Constraints ==========================
Fail [End Next Year, Thing = Water, Place = Eritrea]: 0%. "Not get worse."
============================ Targets ============================
Wish [Thing = Water, Time = Next Decade, Place = Africa]: +20% <- Pan African Council Policy.
Goal [Time = After Five Years, Place = <our local community>, Thing = Water]: < 5%.

Not very impressive, maybe I had better find another, more specific, scale of measure? Maybe use a set of Scales?

Here is an example of a more-specific Scale:

EXAMPLE Scale: % change in water pollution degree as defined by UN Standard 1026.

Here is an example of some alternative and more-specific set of Scales for the 'Environmentally Friendly' example:

EXAMPLE Environmentally Friendly:
Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things, which does not reduce the protection already present in nature.
Air: Scale: % of days annually when <air> is <fit for all humans to breath>.
Water: Scale: % change in water pollution degree as defined by UN Standard 1026.
Earth: Scale: Grams per kilo of toxic content.
Predators: Scale: Average number of <free-roaming predators> per square km, per day.
Animals: Scale: % reduction of any defined [Living Creature] who has a defined [Area] as their natural habitat.

'Environmentally Friendly' is now defined as a complex attribute, because it consists of a number of elementary attributes: {Air, Water, Earth, Predators, Animals}. A different scale of measure defines each of these elementary attributes. Using these Scales we can add Meters, benchmarks, constraints and target levels to describe exactly how Environmentally Friendly we want to be.

Level of Specification Detail

How much detail you need to specify, depends on what you want control over and how much effort it is worth. The basic paradigm of Planguage is you should only elect to do what pays off for you. You should not build a more detailed specification than is meaningful in terms of your project and economic environment. Planguage tries to give you sufficient power of articulation to control both complex and simple problems. You need to scale up, or down, as appropriate. This is done through common sense, intuition, experience and organizational standards (reflecting

Love's Many Attributes

- Trust
  - Truthfulness
  - Broken Appointments
  - Late Appointments
  - Gossiping to Others
- Respect
- Friendship
- Sharing
- Attention
- Understanding
- Support
- Care
- Comfort
- Kissed-ness
- Sex
- Passion
- *other attributes?*

Love.Trust.Truthfulness:
Ambition: No Lies.
Scale: Average Black Lies/Month.
Meter: Confidential Log of Lies.
Past Lies: Past [Ex-Spouse, Two Years Ago]: 42.
Goal [Current Spouse, This Year]: (Past Lies)/2.

Black Lies: Defined As: Non-White Lies.

Figure 5.2
Love is a many-splendored thing! Another example of decomposing a complex subject into its component attributes. This is from a classroom exercise, which was done in two stages. First, we decomposed the complex concept, 'Love' into many aspects. Then we took one attribute at random to see if a reasonable quantified specification could be achieved.

experience). But, if in doubt, go into more detail. History says we have tended in the past to specify too little detail about requirements. The result consequently has often been to lose control, which costs a lot more than the extra investment in requirement specification.

5.3 Language Core: Scale Definition

This section builds on the specification ideas presented in Chapter 4. It discusses the specification of Scales with qualifiers.

Specifying Scales

The Central Role of a Scale within Scalar Attribute Definition

A scale of measure (Scale) is the heart of a scalar specification and essential to support all the targets, constraints and benchmarks. The specified Scale of an elementary scalar attribute is used (*reused!*) within all the scalar parameter specifications of the attribute (that is, within all the Goal, Budget, Stretch, Wish, Fail, Survival, Past, Record and Trend parameters).

Each time a scalar parameter is specified, the Scale dictates what has to be defined. And then, later, each time a scalar parameter definition is read, the Scale 'interprets' its meaning. So the Scale is truly central to the definition of any scalar parameter. Well-defined scales of measure are well worth the small investment to define and refine them.

Specifying Scales using Qualifiers

The scalar attributes (performance and resource) are best measured in terms of defined conditions (for example, specific times and places). If we fail to do this, they lose meaning. People wrongly guess other conditions than you intend, and cannot relate their experiences and knowledge to your numbers. If we don't get more specific by using qualifiers, then performance and resource continue to be vague concepts and there is ambiguity (which times? which places? which events?).

Further, it is important that the set of *different* performance and resource *levels* for different defined conditions are identified. It is likely that the levels of the performance and resource requirements will differ across the system depending on such things as time, location, role and system component.

Decomposing complex performance and resource ideas, and finding market-segmenting qualifiers for differing target levels, is a key method of competing for business.

Embedded Qualifiers within a Scale: A Scale specification can set up useful qualifiers by declaring embedded scale qualifiers, using the format 'defined [<qualifier>]'. It can also declare default qualifier values that apply by default if not overridden, 'defined [<qualifier>: default: <User-defined Variable or numeric value>]'. For example, [. . . default: Novice].

Additional Qualifiers: However, embedded scale qualifiers should not stop you adding any other useful additional qualifiers later, as needed, during specification. But, if you do find you are adding the same type of parameters in almost all specifications, then you might as well design the Scale to include those qualifiers. A Scale should be built to ensure it forces the user to define the critical information needed to understand and control a critical performance or resource attribute.

Here is an example of how user-defined terms (that is, additional qualifiers) can make a quality more specific. Note also, how a requirement can be made conditional upon an event. If the event is not true, the requirement does not apply.

First, some *basic definitions* are required:

EXAMPLE Assumption A:
Basis [This Financial Year]: Norway is still not a full member of the European Union.

EU Trade:
Source: Euro Union Report [EU Trade in Decade 2000–2009].

Positive Trade Balance:
State [Next Financial Year]: Norwegian Net Foreign Trade Balance has Positive Total to Date.

Now we apply those definitions below:

EXAMPLE Quality A:
Type: Quality Requirement.
Scale: % of Goods Delivered, by <value>, which are Returned for Repair or Replacement by Consumers.
Meter [Development]: Weekly samples of 10, [Acceptance]: 30 day sampling at 10% of representative cases, [Maintenance]: Daily sample of largest cost case.
Fail [European Union, Assumption A]: 40% <- European Economic Members.
Goal [EU and EEU members, Positive Trade Balance]: 50% <- EU Trade.
The Fail and the Goal requirements are now defined partly with the help of qualifiers. The Goal to achieve 50% (or more, is implied) is only a valid plan if 'Positive Trade Balance' is true. The Fail level requirement of 40% (or worse, less, is implied) is only valid if 'Assumption A' is true.

5.4 Rules: Scale Definition

Tag: Rules.SD.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: Rules for Scale Definition.

Note: These rules are concerned with the use of scales of measure and also specification of scalar parameters, including specification of numeric values. They complement Rules.SR.

Base: These rules are to be used in addition to the rules for Scalar Requirement Specification (Rules.SR).

R1: **Standard**: The Scale and/or Meter must, wherever possible, be derived from a standard version (held in named files or referenced sources) and the standard shall be source referenced in the specification. For example, Scale: . . . <- Corporate Scale 1.2.

R2: **Notify Owner**: If a Scale or Meter is not standard, a notification must be sent to the appropriate Library Owner to inform them about the availability of this new case. "Note sent to <Library Owner>" will be included as a specification comment to confirm this act.

R3: **Scale Definition**: Each scale definition in a specification is part of an elementary attribute (that is, the associated elementary requirement definition must have a unique tag, and appropriate set of parameters, such as Past and Goal). The scale definition must define the units of measure so that benchmarks, constraints and targets can be set clearly and consistently.

R4: **Elementary Attribute**: An elementary attribute must only have one Scale.

R5: **Differentiate**: A distinction will be made, by using qualifiers, between those system components which must have significantly higher performance levels than others, and components which do not require such levels. "The most ambitious level [across an entire system] can cost too much."[1]

EXAMPLE Goal [Operating System Core]: 99.98%, [Online Internet Components]: 99.90%, [Offline Components]: 99%.

R6: **Uncertainty**: Whenever there is known uncertainty in the precise level for a specified numeric value, its upper and lower boundaries should be explicitly stated. Expressions, such as {60 ± 20, 60 to 80, 60?, 60??}, can be used.

R7: **Scalar Priority**: No artificial 'weights'. Use scalar priority. The relative 'static' (initial) priority of a scalar requirement (its 'claim on limited resources) is initially given by means of the target and constraint statements {Goal, Stretch and Wish, Fail and Survival levels} and, also by the complementary information given by qualifiers, Source and Authority statements. It is unnecessary and 'corrupting' to add any other priority information (such as weights or relative priority).

The final real 'dynamic' priority of meeting a scalar requirement is a matter for systematic engineering tradeoff later, when the total real impacts and costs of design ideas are better understood during design analysis or system development.

(*Note: Function requirements can however state 'simple priority' directly. They have no scalar mechanisms for determining priority based on unfulfilled Goal attainment. See Rules.FR:R5: Function Priority.*)

[1] I once participated in 'saving' a German telecoms project, which had run about 3,000 work years over financial budget and two to three calendar years late, mainly as a result of applying the highest quality levels across the entire system (in fact, only the core software warranted such levels).

Table 5.1 Examples of Scales of Measure.

Performance	*Effect of Change in Performance*	*Scale of Measure*
Customer Satisfaction	Fewer letters of complaint	Number of letters complaining about a defined [Product] received within a defined [Time Period]
Customer Satisfaction	Fewer returned goods	Percentage of defined [Product] returned within defined [Time Period after Purchase] with defined [Customer Issue]
Environmentally Friendly	Improved rating as measured on international standard	Number of defined [Product Type] failing defined [Test] within a defined [Time Period]
User-friendly	Fewer errors made	Percentage of defined [Transaction Type] with defined [Error] input by defined [User Type]
User-friendly	Faster time for completion of transactions	Time in minutes for a defined [Transaction] to be carried out to <satisfactory> completion
Restful Ambience	Calming, relaxing effect	Percentage of users of defined [User Type] agreeing that defined [Room Space] was <restful>
Reliability	Fewer breakdowns	Mean Time Between Repair (MTBR)
Staff Satisfaction	Lower rate of staff turnover	Number of staff of defined [Job Description Response]
Predictability	Less variance in time to initial response	Percentage of service calls of defined [Service Type] exceeding <initial response> within defined [Time Period]

5.5 Process Description: Scale Definition

Process: Scale Definition

Tag: Process.SD.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: Determining a Scale of Measure.

Note: The procedure steps cannot simply be done sequentially. Iteration is needed to evolve realistic scales of measure.

Entry Conditions

E1: The Generic Entry Conditions apply. Input documentation includes contracts, marketing plans, product plans and the requirement specification. The relevant rules should also be available: the generic specification rules (Rules.GS), the requirement specification rules (Rules.RS), the rules for scalar requirement specification (Rules.SR) and, the rules for scale definition (Rules.SD).

E2: Do not enter this procedure if company files or standards already have adequate quantification devices. Preferably use the existing Scales and Meters found in the standards' libraries.

Procedure

P1: Ensure that you have derived an elementary attribute (from a complex requirement), and that you are not trying to use a complex requirement, which needs decomposition into its elementary attributes. (*Trying to find a single Scale for a complex (multi-Scale) requirement doesn't work well. It is usually the cause of trouble when people fail to find a suitable Scale.*)

If you find you do indeed have a complex requirement, then decompose it and try to find Scales for its components. You might well find that further (second-level and more) decomposition is required!

P2: Ensure that the elementary attribute that you are developing a Scale for has a suitable tag and a Gist or Ambition parameter that adequately describes the concept in outline terms.

P3: Using the Gist or Ambition, analyze how a 'change' of degree in the scalar attribute level would be expressed. What would a user experience or perceive? For some examples, see Table 5.1, 'Examples of Scales of Measure'.

Sometimes you can keep things simple, and 'make do', by controlling the details at a higher level of abstraction:

- *by deciding to use one dominant Scale only, and consciously ignoring the potential other scales.*
- *by aggregating several scales of measure to express one summary scale of measure.*
- *by defining a complex attribute as the 'set' of other Scales and definitions.*

P4: Specify the critical [time, place, event] qualifiers to express different benchmarks, constraints and target levels.

P5: If there is no appropriate standard Meter (or test), start working on a Meter. Try to imagine a practical way to measure things along the Scale, or at least sketch one. Try thinking about any measures that are currently being carried out (this could even help you start developing ideas for scales of measure). Also, think about whether any current system could be modified, or have its settings changed, to perform additional measurement.

P6: Try out the Scale. Define some reference points from the past (*benchmarks*) and then, on the basis of benchmarks, specify future requirements (*targets* and *constraints*).

P7: Repeat this process until you are satisfied with the result. Try to get approval for your Scale from some of the stakeholders. Does it quantify what they really care about?

P8: Consider putting embedded parameters into the Scale definition. Rationale: To enable a Scale to be reused both within a project and in other projects.

EXAMPLE Scale: Time needed to do defined [Task] by defined [User] in defined [Environment]. Goal [Task = Get Number, User = <Novice>, Environment = <Noisy>]: 10 minutes.

P9: Once you have developed a useful Scale, ensure it is made available for others to use (on your intranet, or a web site, or in course materials, or your 'personal' glossary of Scales[2]). Offer the Scale to the owner of the 'Scales' library within your organization.

Exit Conditions

X1: The Generic Exit Conditions apply.

X2: Alternatively, consensus is obtained on trying out the Scale in practice, and exit condition X1 is temporarily waived.

Rationale [X2, Tryout]: The intent being to gain experience, or to obtain opinions concerning the quantification, so it can be refined ready for <official use>.

5.6 Principles: Scale Definition

1. **The Principle of 'Defining a Scale of Measure'**
   If you can't define a scale of measure, then the goal is out of control.
   Specifying any critical variable starts with defining its scale.
2. **The Principle of 'Quantification being Mandatory for Control'**
   If you can't quantify it, you can't control it.[3]
   If you cannot put numbers on your critical system variables, then you cannot expect to communicate about them, or to control them.

[2] See http://www.Gilb.com/. It is easy to find examples of scales by searching the web, for example, search for 'Usability metrics'.
[3] Paraphrasing a well-known old saying.

3. **The Principle of 'Scales should control the Stakeholder Requirements'**
   Don't choose the easy Scale, choose the powerful Scale.
   Select scales of measure that give you the most direct control over the critical stakeholder requirements. Choose the Scales that lead to useful results.

4. **The Principle of 'Copycats Cumulate Wisdom'**
   Don't reinvent Scales anew each time – store the wisdom of other Scales for reuse.
   Most scales of measure you will need will be found somewhere in the literature or can be adapted from existing literature.

5. **The Cartesian Principle**
   Divide and conquer said René – put complexity at bay.
   Most high-level performance attributes need decomposition into the list of sub-attributes that we are actually referring to. This makes it much easier to define complex concepts, like 'Usability', or 'Adaptability,' measurably.

6. **The Principle of 'Quantification is not Measurement'**
   You don't have to measure in order to quantify!
   There is an essential distinction between quantification and measurement. "I want to take a trip to the moon in nine picoseconds" is a clear requirement specification without measurement."
   The well-known problems of measuring systems accurately are no excuse for avoiding quantification. Quantification allows us to communicate about how good scalar attributes are or can be – before we have any need to measure them in the new systems.

7. **The Principle of 'Meters Matter'**
   Measurement methods give real world feedback about our ideas.
   A 'Meter' definition determines the quality and cost of measurement on a scale; it needs to be sufficient for control and for our purse.

8. **The Principle of 'Horses for Courses'**[4]
   Different measuring processes will be necessary for different points in time, different events, and different places.[5]

9. **The Principle of 'The Answer always being 42'**[6]
   Exact numbers are ambiguous unless the units of measure are well-defined and agreed.
   Formally defined scales of measure avoid ambiguity. If you don't define scales of measure well, the requirement level might just as well be an arbitrary number.

[4] 'Horses for courses' is a UK expression indicating something must be appropriate for its use.
[5] There is no universal static scale of measure. You need to tailor them to make them useful.
[6] The concept of the answer being 42 was made famous in Douglas Adams, *The Hitchhiker's Guide to the Galaxy*, Macmillan, 1979, ISBN 0-330-25864-8.

10. **The Principle of 'Being Sure About Results'**
    If you want to be sure of delivering the critical result – then quantify the requirement.
    Critical requirements can hurt you if they go wrong – and you can always find a useful way to quantify the notion of 'going right.'

5.7 Additional Ideas: Generic Hierarchies for Scalar Attributes

You can decompose many scalar attributes into arbitrarily large or small sets of more specific 'elementary' scalar attributes. The selection of exactly which elementary attributes to define is a practical matter of knowing your domain well enough to decide which of them will give you best control over your critical success factors. At best we make reasonable guesses with some effort to begin with. Then we learn some hard lessons, usually about what we forgot to exercise control over.

Having said this, we have found that templates for performance and resource/cost attributes are helpful to most people. So, we will give some basic performance attributes in this section. Remember, in any real system they will need to be used *selectively*: they will need to be *tailored* to your local purpose. (*See Figure 5.4 for an overview of these attribute definitions.*)

Note all these template ideas build upon the templates originally presented in Gilb (1988 Chapter 19).

They are organized into multilevel hierarchies of attributes. This does not imply that any one hierarchical organization is best or correct. But they are useful. The essential idea is to get control over those elementary attributes that determine your success or failure. A flat list of the right ones works as well as any hierarchy. Hierarchies are mainly useful groups for human convenience, but are not a reality for the system.

Hierarchy of Performance

Performance: 'Useful values deliverable to stakeholders.'
Includes: {Quality, Resource Savings, Workload Capacity}.

1. **Quality**: 'How well a system performs.'
   Includes: {Availability, Adaptability, Usability, Other}.

   1.1 **Availability**: 'The readiness of a system to do its work.'
   Gist: Availability is the measure of how much a system is usefully (not merely technically) available to perform the work that it was designed to do.

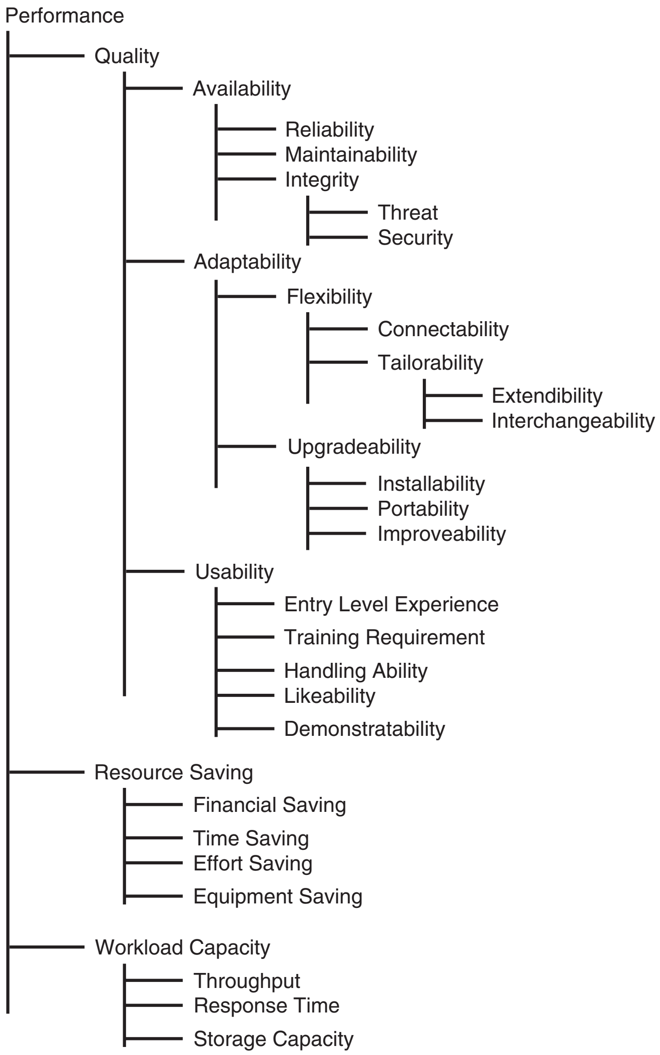


Figure 5.3
One decomposition possibility for performance attributes with emphasis on the detail of the quality attributes.

Availability: Type: Elementary Quality Requirement.
Scale: % of defined [Time Period] a defined [System] is available for its defined [Tasks].

Availability: Type: Complex Quality Requirement.
Includes: {Reliability, Maintainability, Integrity}.

1.1.1 **Reliability**: 'A system performs as it is intended.'
Gist: Reliability is a measure of the degree to which a system performs as it was designed to do, as opposed to doing something else (like producing a wrong answer or providing

no answer). Definitions of Reliability will therefore vary according to the definition of what the system is supposed to do. In general, if a system is in an unreliable state then it is 'unavailable' for its intended work tasks.

Scale: Mean time for a defined [System] to experience defined [Failure Type] under defined [Conditions].

1.1.2 **Maintainability**: 'Resource required to repair an unreliable system.'
Gist: Maintainability is a measure of how quickly an unreliable system can be brought to a reliable state. In general, this covers not only the actual repair of the fault, but also recovery from any effects of the fault and, quality control and test of the repair.
Conventionally, maintenance is concerned with the process of fault handling, rather than for improvement of a faultless system. However, the difference between what is a fault and what is a system improvement can be subjective! (*See also later definition for 'Adaptability'*.)

Maintainability: Type: Elementary Quality Requirement.
Scale: Mean time to carry out a defined [Type of Repair] to a defined [System] using defined [Repair Method] under defined [Conditions].

1.1.3 **Integrity**: 'The ability of the system to survive attack.'
Gist: Integrity is a measure of the confidence that the system has suffered no harm: its security has not been breached and, its use has resulted in no 'corruption' or impairment to it. An attack on the Integrity of a system can be accidental or intentional. The Integrity of a system depends on the frequency of *threat* to it and the effectiveness of its *security*.

Integrity: Type: Elementary Quality Requirement.
Scale: Probability for a defined [System] to achieve defined [Coping Action] with defined [Attack] under defined [Conditions].
Coping Action: {detect, prevent, capture}.
Integrity: Type: Complex Quality Requirement.
Includes: {Threat, Security}.

1.2 **Adaptability**: 'The efficiency with which a system can be changed.'
Gist: Adaptability is a measure of a system's ability to change. Since, if given sufficient resource, a system can be changed in almost any way, the primary concern is with the amount of resources (such as time, people, tools and finance) needed to bring about specific changes (the 'cost').

Maintainability:
Type: Complex Quality Requirement.
Includes: {Problem Recognition, Administrative Delay, Tool Collection, Problem Analysis, Change Specification, Quality Control, Modification Implementation, Modification Testing {Unit Testing, Integration Testing, Beta Testing, System Testing}, Recovery}.

Problem Recognition:
Scale: Clock hours from defined [Fault Occurrence: Default: Bug occurs in any use or test of system] until fault officially recognized by defined [Recognition Act: Default: Fault is logged electronically].
Administrative Delay:
Scale: Clock hours from defined [Recognition Act] until defined [Correction Action] initiated and assigned to a defined [Maintenance Instance].
Tool Collection:
Scale: Clock hours for defined [Maintenance Instance: Default: Whoever is assigned] to acquire all defined [Tools: Default: all systems and information necessary to analyze, correct and quality control the correction].
Problem Analysis:
Scale: Clock time for the assigned defined [Maintenance Instance] to analyze the fault symptoms and be able to begin to formulate a correction hypothesis.
Change Specification:
Scale: Clock hours needed by defined [Maintenance Instance] to fully and correctly describe the necessary correction actions, according to current applicable standards for this.
Note: This includes any additional time for corrections after quality control and tests.
Quality Control:
Scale: Clock hours for quality control of the correction hypothesis (against relevant standards).
Modification Implementation:
Scale: Clock hours to carry out the correction activity as planned. "Includes any necessary corrections as a result of quality control or testing."
Modification Testing:
Unit Testing:
Scale: Clock hours to carry out defined [Unit Test] for the fault correction.
Integration Testing:
Scale: Clock hours to carry out defined [Integration Test] for the fault correction.
Beta Testing:
Scale: Clock hours to carry out defined [Beta Test] for the fault correction before official release of the correction is permitted.
System Testing:
Scale: Clock hours to carry out defined [System Test] for the fault correction.
Recovery:
Scale: Clock hours for defined [User Type] to return system to the state it was in prior to the fault and, to a state ready to continue with work.

Source: The above is an extension of some basic ideas from Ireson, Editor, Reliability Handbook*, McGraw Hill, 1966* (Ireson 1966).

Figure 5.4
A more detailed view of Maintainability.

Adaptability: Type: Elementary Quality Requirement.
Scale: Time needed to adapt a defined [System] from a defined [Initial State] to another defined [Final State] using defined [Means].

Adaptability: Type: Complex Quality Requirement.
Includes: {Flexibility, Upgradeability}.

1.2.1 **Flexibility**:
Gist: This concerns the 'in-built' ability of the system to adapt or to be adapted by its users to suit conditions (without any fundamental system modification by system development).
Type: Complex Quality Requirement.
Includes: {Connectability, Tailorability}.

1.2.1.1 **Connectability**: '*The cost to interconnect the system to its environment.*'
Gist: The support in-built within the system to connect to different interfaces.

1.2.1.2 **Tailorability**: '*The cost to modify the system to suit its conditions.*'
Type: Complex Quality Requirement.
Includes: {Extendibility, Interchangeability}.

1.2.1.2.1 **Extendibility**:
Scale: The cost to add to a defined [System] a defined [Extension Class] and defined [Extension Quantity] using a defined [Extension Means].
"In other words, add such things as a new user or a new node."
Type: Complex Quality Requirement.
Includes: {Node Addability,
Connection Addability,
Application Addability,
Subscriber Addability}.

1.2.1.2.2 **Interchangeability**: '*The cost to modify use of system components.*'
Gist: This is concerned with the ability to modify the system to switch from using a certain set of system components to using another set.
For example, this could be a daily occurrence switching system mode from day to night use.

1.2.2 **Upgradeability**: '*The cost to modify the system fundamentally; either to install it or change out system components.*'
Gist: This concerns the ability of the system to be modified by the system developers or system support in planned stages (as opposed to unplanned maintenance or tailoring the system).
Type: Complex Quality Requirement.
Includes: {Installability, Portability, Improveability}.

1.2.2.1 **Installability**: '*The cost to install in defined conditions.*'
This concerns installing the system code and also, installing it in new locations to extend the system

coverage. Could include conditions such as the installation being carried out by a customer or, by an IT professional on-site.

1.2.2.2 **Portability**: '*The cost to move from location to location.*'
Scale: The cost to transport a defined [System] from a defined [Initial Environment] to a defined [Target Environment] using defined [Means].
Type: Complex Quality Requirement.
Includes: {Data Portability,
Logic Portability,
Command Portability,
Media Portability}.

1.2.2.3 **Improveability**: 'The cost to enhance the system.'
Gist: The ability to replace system components with others, which possesses improved (function, performance, cost and/or design) attributes.
Scale: The cost to add to a defined [System] a defined [Improvement] using a defined [Means].

1.3 **Usability**: '*How easy a system is to use.*'
Scale: Speed for defined [Users] to correctly accomplish defined [Tasks] when given defined [Instruction] under defined [Circumstances].
Note: This is a generic scale for Usability, which you can use if you want to simplify matters and deal with Usability at an elementary level. It is however more usually declared as 'complex' and then defined in a more specific manner; for example, by using the sub-attributes below. There are of course, many different possible decompositions of Usability.
Type: Complex Quality Requirement.
Includes: {Entry Level Experience, Training Requirement, Handling Ability, Likeability, Demonstratability}.

1.3.1 **Entry Level Experience**:
Scale: The defined [Level of Knowledge] required to receive training or to use a defined [System].

1.3.2 **Training Requirement**:
Scale: The degree of training required for a defined [User Type] to achieve a defined [Degree of Proficiency] with a defined [System].

1.3.3 **Handling Ability**:
Scale: A defined [Degree of Proficiency] with a defined [System] by a defined [Class of User].

1.3.4 **Likeability**:
Scale: The degree to which defined [Users] declare that they are pleased with defined [Aspects] of a defined [System].

1.3.5 **Demonstrability**:
Type: Complex Quality Requirement.
Includes: Elementary Quality Requirement {Customer Self-Demonstrability, Sales Demonstrability}.

Some Alternative Models for Usability:
The point of these three alternative models to the basic Usability model (above) is to emphasize that there is NOT one 'correct model.' All major projects need highly tailored models. I also want to show some specific instances of Usability sub-scales as a checklist or stimulant to readers when building their own models.

EXAMPLE **Usability**:
Type: Complex Quality Requirement.
Includes: {Entry Level Experience, Training Requirement, Handling Ability, Likeability, Demonstrability}. "Only one of the many possible decompositions of Usability."
Demonstrability:
Type: Complex Quality Requirement.
Includes: Type: Elementary Quality Requirement {Customer Self-Demonstrability}.
Customer Self-Demonstrability:
Ambition: Ability of Customer to solo self-demonstrate a Product is to be <high>.
Scale: Probability of <successful completion> of self-demonstration within one hour.
Past [Last Year, All Products]: < 5%.
Fail: 90% to 95% <- Corporate Quality Policy.
Goal: 95%.

EXAMPLE **Usability**:
Type: Complex Quality Requirement.
Device Swapability:
Scale: Minutes to swap over a defined [Input Device or Output Device].
Training Need: Scale: Hours in <training mode> until capable of defined [Tasks].
User Productivity: Scale: % User Time lost due to Product Fault or <Bad Design>.
User Error Rate: Scale: % of User Actions, which they correct or change.
User Minimum Qualification Level:
Scale: Average % of correct answers to a defined [Qualifying Test] by a defined [User Type].
Userlessness: Scale: % of Tasks, which can run <unattended>.
Coherence:
Scale: % of User Interface Elements, which are perceived as consistent with our Product Image.

User Opinion:
Scale: % of defined [User Type] who express <positive feeling> after using defined [Product Component(s)].
Customer Self-Demonstratability:
Scale: % Probability of successful <self demonstration> of defined [Product or Product Component] by defined [User Type] within defined [Time Span] of attempt to use it.

EXAMPLE **Usability**:
Type: Complex Quality Requirement.
Includes: Type: Elementary Quality Requirement {Entry Conditions, Training Requirement, Computer Familiarity, Web Experience Level, Productivity, Error Rate, Likeability, Intuitiveness, Intelligibility}.
Entry Conditions:
Scale: <Grade Level of User>.
Training Requirement:
Scale: Time needed to read <any instructions> or get <any help> in order to perform defined [Tasks] successfully.
Computer Familiarity:
Scale: Years of <experience with computers>.
Web Experience Level:
Scale: Years of <experience with using the web>.
Productivity:
Scale: Ability to correctly produce defined [Work Units: Default: Completed Transactions].
Error Rate:
Scale: Number of Erroneous Transactions requiring correction each <session>.
Likeability:
Scale: Option of <pleasure> on using the system on scale of -10 to $+10$.
Intuitiveness:
Scale: Probability that a defined [User] can intuitively figure out how to do a defined [Task] correctly (without any errors needing correction).
Intelligibility:
Scale: Probability in % that a defined [User] will correctly interpret defined [Messages or Displays].

2. **Resource Savings**:
Gist: How much resource savings a new system produces compared to some defined benchmark system.
Type: Complex Performance Requirement.
Includes: {Financial Saving, Time Saving, Effort Saving, Equipment Saving}.

- **Financial Saving**: "Financial Cost Reduction"
  Scale: Net Financial Saving planned or achieved compared to a defined [Benchmark Amount].
- **Time Saving**: "Processing Time Reduction, Elapse Time Reduction, Time To Market"

Scale: Net Time Saving planned or achieved compared to a defined [Benchmark Amount].

- **Effort Saving**: "Reduction in the Person-Hours required"
  Scale: Net Effort saving planned or achieved compared to a defined [Benchmark Amount].

- **Equipment Saving**: "Includes room space!"
  Scale: Net Space saving planned or achieved compared to a defined [Benchmark Amount].

3. **Workload Capacity**: "The raw ability of the system to perform work."
   Type: Complex Performance Requirement.
   Includes: {Throughput, Response Time, Storage Capacity}.

- **Throughput**:
  Gist: Throughput is a measure of the ability of the system to process work. For example, the average number of telephone sales orders, which can be dealt with by an experienced telephone sales operator, in an hour.
  Scale: The average quantity of defined [Work Units], which can be successfully handled in a defined [Time Unit].

- **Response Time**: "Retrieval Timing, Transaction Timing"
  Scale: The mean average speed to perform a defined [Reaction] on receiving a defined [Impulse].

- **Storage Capacity**: "The ability of the system to increase in size"
  Gist: This is the capability of a component part of the system to store units of some defined kind. For example, number of registered users, lines of code, photographs and boxes.
  Scale: The capacity to store defined [Units] under defined [Conditions].

5.8 Further Example/Case Study: Scale Definition

This is part of a quality definition done for the airborne command and control system, which was discussed previously in Section 3.8. It is a first draft (there are lots of things to be refined later) and it is only a sample of the requirement specification we actually worked out. We chose to work on 'Usability' as it was defined as 'the key competitive system quality'. This system is now operational.

EXAMPLE

Usability:
Ambition: Operator ease of learning & doing tasks under <all conditions> should be maximum possible ease & speed of performance with minimum training & minimum possibility of <unchecked error(s)>.

Usability.**Intuitiveness**:
Ambition: High probability that an operator will within a specified time from deciding the need to perform a specific task (without reference to handbooks or help facility) find a way to accomplish their desired task.
Scale: Percentage Probability that a defined [Individual Person: Default: Trained Operator] will find a way to perform a defined [Task Type] without reference to any written instructions, other than the help or guidance instructions offered by the immediate system screen (that is, no additional paper or on-line system reference information), within a defined [Time Period: Default: Within one second from deciding that it is necessary to perform the task].
Comment [Intuitiveness:Scale]: "I'm not sure if one second is acceptable or realistic, it's just a guess" <- MAB.
Meter: To be defined. Not crucial this 1st draft <- TG.
Past [System R]: 80%? <- LN.
Record [Mac User Interface]: 95%? <- TG.
Fail [**Trained Operator, Rare Tasks** [{<1/week, <1/year}]]: From 50% to 90%? <- MAB.
Goal [**Tasks Done** [<1/week (but more than 1/Month)]]: 99%? <- LN,
 [**Tasks Done** [<1/year]]: 20%? <- JB,
 [Turbulence, **Tasks Done** [<1/year]]: 10% ? <- TG.
====================== User Defined Terms ======================
Trained Operator: Defined As: Command and Control Onboard Operator, who has been through approved training course of at least 200 hours duration.
Rare Tasks: Defined As: Types of tasks performed by an Onboard Operator less than once a week on average.
Tasks Done: Defined As: Distinct tasks carried out by Onboard Operator.
==================== =================== ====================
Usability.**Intelligibility**:
Ambition: High ability for an operator to <correctly> interpret the meaning of given information.
Scale: Percentage Probability of <objectively correct> interpretation(s) of a defined [Set of <Inputs>] by a defined [Individual Person: Default: Trained Operator] within a defined [Time Period].
Meter [Acceptance]: Use about 10 Trained Operators, and use about 100 <representative sets of information per operator within 15 minutes?> - MAB.
Comment [Meter]: "Not sure if the 15 minutes are realistic" <- MAB.
Comment [Meter]: "This is a client & contract determined detail" <- MAB.
M1: Past: [XXX, 20 Trained Operators, 300 <data sets>, 30 minutes]: 99.0% <- Acceptance Test Report from XXX, MAB.
Record [XXX]: 99.0%. "None other than XXX known by me" <- MAB.
Fail [First Delivery Step]: 99.0%? <- MAB.
Fail [Acceptance]: 99.5%? <- MAB.
Goal [XXX, 20 Trained Operators, 300 <data sets>, 30 minutes]: 99.9% <- LN.

==================== More User Defined Terms ====================
Acceptance: Defined As: Formal Acceptance Test as defined by our contract with Customer XXX.
First Delivery Step: Defined As: By end of November this year (The results of the first evolutionary result cycle will be integrated into the system and will be producing useful results).

5.9 Diagrams/Icons: Scale Definition

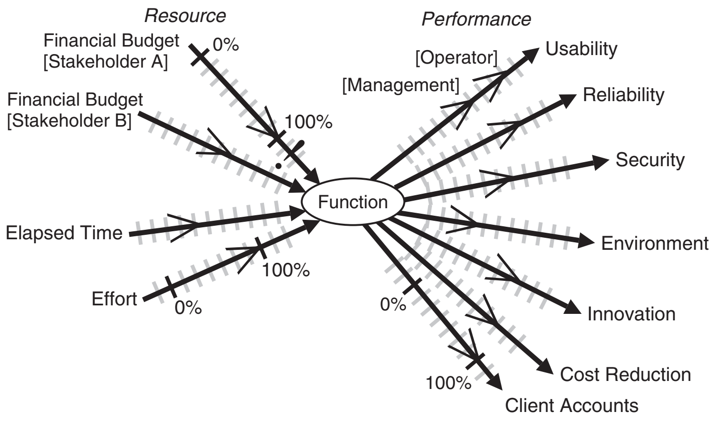


Figure 5.5
A representation of multiple performance and resource attributes showing goal and budget levels respectively. The 'point' of the icon goal and budget symbols indicates the level (reference needs to be made to the Scale to interpret the numeric value). One constraint, a Fail level, is shown on the resource attribute for Financial Budget [Stakeholder A]. The lines of the arrows represent the scales of measure (divisions along the scales are also marked).

5.10 Summary: Scales of Measure

Quantification of all performance and resource concepts must be taken seriously. Ideally, you need to have a corporate policy that all such ideas will be expressed quantitatively at all times. Nothing less will satisfy 'the need to be the best' in a fast-changing competitive world.

Here is a summary of the key ideas about *scales of measure*:

- you can and should always define a scale of measure for any system critical variable performance or resource attribute

- defining a scale of measure is a teachable practical process
- specification of a scale can be done using embedded qualifiers, which makes it more immediately powerful and also reusable in other projects
- most scales of measure are tailored variations of a generally applicable set of scales (like Usability and Maintainability). Once you have learned the general set, it becomes much easier to generate useful scales as needed for variations
- qualities do not have to be expressed 'qualitatively' (for example, using words like 'high security') – they should be quantified for serious Competitive Engineering
- an organization should make a library of useful scales of measure for its area of interest
- really good scales of measure are tailored – truly general scales (like 'volts') are not likely to be what you need for best competitiveness
- scales of measure in requirements are the foundation of understanding any design or architecture impact on that requirement – both when it is being considered, and then when it is being implemented in practice.

If you think you know something about a subject, try to put a number on it. If you can, then maybe you know something about the subject. If you cannot then perhaps you should admit to yourself that your knowledge is of a meagre and unsatisfactory kind.

Lord Kelvin, 1893

Chapter
6

Resources, Budgets and Costs

Costs of Solutions

Glossary Concepts

Resource
Budget
Cost

6.1 Introduction

You only get what you pay for.

Folk wisdom

A system designer tries to meet the specified system requirements by identifying value-producing design ideas (solutions). At the same time as looking for function and performance, the designer must also consider the *resources needed*, specifically respecting any *constraints placed on resource usage*.

Relationships amongst Resources, Budgets and Costs

Resources are the inputs, or the 'fuel,' for a system. They are needed to produce the system's performance attributes. They are analogous to the capital expense, air and fuel needed for a car engine (function attribute is to provide power) to deliver the engine's performance attributes.

Resource requirements specify *how much* we plan to use of a limited resource to bring about change (new systems, improved systems) and/or to operate a system. Resource requirements are also known as *budgets*.

Stakeholders' *resources* pay all the real project and system *costs*. In other words, *costs* are the actual consumption of resources. Resource requirements are therefore sometimes termed *cost requirements* (*or cost budgets*).

The term 'resources' is used here in the broadest sense of that word. It covers money, time, people, space and any other 'currency' with which we pay for system changes and the operational system.

Stakeholder Requirements and Resources

Projects exist *primarily* to deliver stakeholder performance requirements. A system's *functions* are probably already in place, and may well have been for ages in earlier generations of the system, but the projected performance outputs (qualities, workload capacities and/or resource savings) of the system are probably not satisfactory – or they will not be in the future. That is what puts you in the 'business of change,' in other words 'creates your project.'

Any project sponsor has limited resources, and is faced with alternative ways to use them. Projects must control costs, or they will either exceed their project sponsors' capability for providing resources or be seen as a less attractive (read 'less profitable') investment for those resources.

For most of today's projects, controlling cost (resource expenditure) is quite a juggling act: you have to balance and trade off performance against budgets. Your stakeholders want a better system, but not at too high a cost. They can usually specify a 'budget' for what they are willing to pay for each system improvement, based on their knowledge of their current system and their competitors' systems. Of course, their budgets may or may not be realistic!

There may also, in practice, actually be *real and absolute limits* on their budgets, which are in no way just 'hopeful plans'. These limits will more severely restrict the amount of resource that can be made available.

'Limited resources' means that either there are necessary economic *limits (it would not be profitable to spend more, or other projects need these resources more) or* finite *availability (there is really no more resource available at all).*

To complicate matters further: it could also be the case that some of your competitors are *also* willing to provide your stakeholders with improvements. Maybe, only if you bid the *lowest-cost* solution for the defined system performance levels, will you get any development business whatsoever.

The Relationship between Costs and Performance Delivery

Many, but not all, system performance attributes are directly related to the operational costs of using the system or to the costs of changing the operational system. As examples, think of qualities such as 'Maintainability' and 'Reliability' and workload capacities such as 'Response Time.' To give a specific example, a project might invest some resources to produce a system with a 'higher ease of maintenance.' The resulting system, in *operation*, will have long-term *lower Maintenance Costs* – due to the improved Maintainability attribute level (Scale: <mean time to repair>.) which was the result of a one-off *investment (an implementation cost).*

Ultimately, every system requirement can be viewed in terms of resources. When making decisions about system changes, a stakeholder is merely exercising choice over where resources are to be expended – by choice (now) or by default (later)!

The key point is that there usually is *choice* about where and when resources are expended. To be competitive, not only must a stakeholder consider if an investment bears a clear relationship to producing the required benefits, they must also be sure that the specified 'required benefits' are the 'correct' objectives and that the selected investment is going to give the best available payoff.

Look at the Use of Resources across the Entire System Lifetime

There is no point in narrow cost control. We need 'value for money' control instead. We must learn to balance the use of resources across the entire system lifetime. To give some examples:

- there is little use in simply controlling an implementation project's financial investment, if the result is excessive operational costs for the resulting system, or excessive system retirement costs
- it is no good constraining the time to market, if the consequence is that the product cannot achieve the necessary performance levels for sales on that market
- there is no point in constraining head count on a project only to experience that the consequence is project delays to market, which threaten profitability.

The design engineer must be able to intelligently trade off and balance, to some reasonable degree, all the many performance and cost requirements. To do this, a full set of requirement specifications is required across the entire system lifetime. Otherwise, any tradeoffs will be carried out without knowledge of the 'full picture,' and short-term priorities will tend to dominate.

Numeric Performance Levels Enable Us to Understand the Associated Costs Better

Numeric performance requirement specification, with sufficient precision for purpose, is necessary in order to be able to calculate the costs of achieving the performance levels with any precision. Conventions such as specifying performance levels as 'low,' 'medium,' 'high' and 'extremely high,' will not allow us to exercise reasonable control over costs. For example, availability levels like 99.90%, 99.98% and 99.998%, which can easily have order-of-magnitude cost differentials to achieve, would be impractical to distinguish amongst by merely using such non-numeric terms.

The Cost of Perfection – Beware Infinite Cost Increases

'Perfect quality' does not seem possible in our world and lifetime. The stakeholder would always *like* to have it (the 'Ideal' level), but cannot *ever* afford it in practice. It seems that the 'cost of perfection' is *infinite*

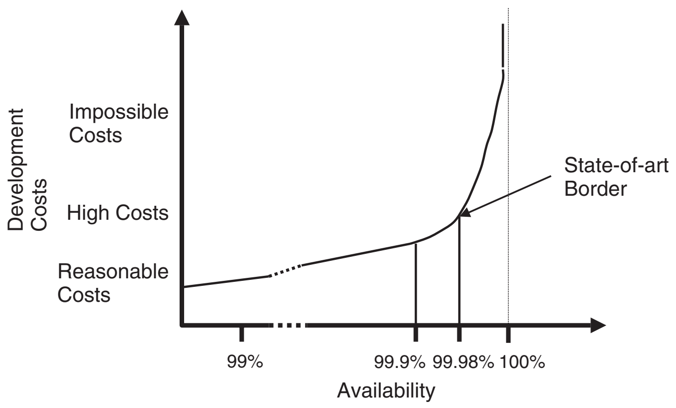


Figure 6.1
As we move any performance level towards nearing perfection, we increase costs dramatically in the direction of infinite costs.

resources. More practically, the costs of performance levels 'nearing perfection' have a nasty tendency to accelerate *towards* infinity. So, as we become more *ambitious* regarding performance, we must become much more *exact* at specifying the performance levels, if we are to hope to understand and control the cost implications.

Further, we must also understand that our systems can be *sensitive* to very small changes in *any* attribute or design specification. These seemingly small changes can give unexpectedly large cost increases, incalculable in advance.

Specify Costs Down to a More Detailed Level – Not Just Total Costs!

We also need to specify the cost requirements in far more detail than people usually do. Not a simplistic 'bottom-line-for-everything' cost budget, but in *detail.* What costs are associated with *every increment* of performance? What costs accompany each increment of function? Estimating and tracking detailed costs will improve our capability of getting feedback early and correcting any situation where the costs are getting 'out of line'. One practical way to view such cost information is by using an Impact Estimation table (IE table) (*see Section 6.8 and Chapter 9, 'Impact Estimation'*).

Accurate Estimation of Costs in Advance is Unlikely for Complex Systems

In advance of building and delivering complex systems (or parts of them), there is no *reliable* way with reasonable *accuracy,* to compute the real, final cost or, to compute the consequential, longer-term cost (Morris 1994). History shows that it is more successful to *stipulate* a reasonable budget amount, and then 'see how much you can get out of it' (MacCormack 2001; Mills 1980). *This means that cost budgets cannot really (and should not) be unilaterally fixed in advance for defined performance requirements.*

Multiple cost budgets and multiple performance goal levels must somehow be set together in some reasonably 'balanced' way. The exact balances amongst them may well be difficult to estimate or know in advance: only the inexperienced believe they can accurately calculate such effects. But we can 'learn as we go' about expected costs in small increments of experience.

Use Design to Cost and Evolutionary Project Management (Evo)

In practice, the best approach to controlling costs for complex systems must be to '*Design to Cost*,' and then to use the Evolutionary Project Management (Evo) method (*see Chapter 10*) and track actual costs.

'Design to Cost' means that you intentionally select designs which *fit within* your committed cost budgets. You may even trade off some marginal performance levels in order to stay within your resource constraints and meet resource targets (budgets). It depends on your priorities. (The alternative is to design for performance alone, and be surprised at the budget overruns!)

Using the Evo method for your project means delivering to your customer or market a succession of improvements in the system's functionality and performance levels. The highest priority improvements must be delivered 'first' (at the earliest opportunities). You must be prepared to learn from the *frequent feedback* from the partial deliveries and to make any *necessary adjustments* in cost budgets. In practice, this is in your interest because, with early warning, you can 'change course' early and so avoid many cost problems.

When, eventually, the budgeted resources do run out – even if you have not delivered all the requirements yet – you can ask, like Oliver Twist, for 'another cup of broth.' If you have been good at delivering value in relation to the resources you have used, then one would expect that your stakeholders would want to keep you in business (the next 'round', at least).

The reader may well find that the ideas of 'Design to Cost,' and of taking an evolutionary approach to costs are strange. But we argue that they are both necessary and possible. Planguage has these approaches in-built in the form of the Impact Estimation and Evolutionary Project Management methods. Even when performance requirements are set at the highest levels, Evolutionary Project Management has a successful past history of being in control of costs and deadlines (for example, the space and military projects in the late 1970s). *(More on this method can be found in Chapter 10, 'Evolutionary Project Management.' More on 'Design to Cost' can be found in Chapter 7, 'The Design Process' and in Chapter 9, 'Impact Estimation.')*

We can control costs if we get early warnings of unexpected costs and we are able to react to these warnings. We must have early, frequent, feedback mechanisms in our planning, our systems engineering and our project management. We can get this degree of control:

- by budgeting resources in *small* (say, 2%) increments
- by *designing* to stay within the budget
- by *reacting to experience* with cost expenditure (changing designs or requirements as far as it is realistic to do so)
- by monitoring a *multiplicity* of resource budgets and a *multiplicity* of performance goals
- by specifying all the *constraints* that apply to the problem, in advance of solving it.

6.2 Practical Example: Resources, Budgets and Costs

Resource Requirement Specifications: Allocation of Resources

We are all familiar with the simplest types of 'resource limitation' specifications: 'the total budget is a million' and 'the deadline is January next year.' There is a real human need for these simple ideas.

However, in order to control and deliver 'within budget', we must take a more sophisticated approach to budget specification. We must, for example, relate resources more carefully to *exactly* what is to be achieved or delivered to stakeholders (the required function and performance attributes), and we must consider the resource constraints. If we fail to do so, then both time and money will run out but we will not have achieved our 'real aims,' which are the function and performance improvements. For example, if only 2% or 20% of the work is accomplished by using 80% of our budget, then we are usually in deep trouble.

Here is an example of a generic financial budget specification, which helps ensure more specific detail:

EXAMPLE

Financial Budget:
Scale: Percentage (%) of total initial Project Money Allocation.
Type: Resource Requirement.
Meter: Project Accounting.
========================== Constraints ==========================
Survival [Final Deadline]: 100% "Must not use more than this by final deadline".
Rationale [Survival]: If >100% we have a loss on this project, and it can be deemed a failure.
Fail [Final Deadline]: 90%. Rationale: This gives us 10% profit.
============================ Targets ============================
Budget [For each 2% of Total Project Calendar Time, If 2% Benefit]: 2% "of total budget. See Scale above."
2% Benefit: Defined As: At least an incremental 2% of the total of all planned performance improvement ('benefit') shall be delivered.
Rationale [2% Benefit]: This Evo approach will give us consistent control and feedback throughout the project, so we can take action early if necessary, to avoid disaster.
Stretch [Final Deadline]: 80%.
Rationale [Stretch]: This gives us 20% profit. "Double the normal."
Notice the subtle distinction between a Survival level (a hard budget constraint level to avoid unacceptable losses), and a Fail level (a softer budget constraint level to avoid some sort of failure or pain). The Budget is the actual required target budget for some degree of success. The final target set, 'Stretch,' is intended as a motivating cost target. Consider how the resulting 'differentiated' project budget plan will differ from the simplest budget maximum specification.

In the example, we are specifying in the Budget level that for every 2% of our budget we had better not plan to use more than 2% of the calendar time budgeted, and we had better plan to deliver corresponding planned performance improvements in measurable increments. I remind the reader that the previous two chapters tried to introduce the notion that performance measures (such as 2% of any planned performance improvement) *can* be specified and measured.

If you want project control, you will insist on doing things on such a 'pay as you go' basis (or even, 'no cure, no pay'). If you let projects spend money, without demanding clearly measurable results, I promise you 'they' will spend your money, take *your* time and be unable to give you *anything* worthwhile in return. On several occasions, I have investigated very large projects, in the UK, Sweden and Germany, which have managed to consume hundreds of millions of dollars without delivering a single solution of any value to any stakeholder. Take steps to ensure this doesn't happen on your project!

EXAMPLE Engineering Hours:
Gist: To help ensure resource usage is balanced with the business progress and value by controlling the allocation of engineering hours to different stages and types of work.
Ambition: Low resource-use in beginning, more as value increases.
Type: Resource Requirement [Engineering Work-Hours].
Scale: % of total Engineering Work-Hours allocated.
Budget [Early Pilot Trials]: 10%,
[Domestic Deliveries to Contracts]: 10% <- Marketing Plan 6.8,
[From Next Year, Domestic Deliveries, Wholesalers]: 20%,
[European Deliveries, Contracts [At least 10 signed], If Authority Given]: 30%,
[European Deliveries, Wholesalers [1 in each country]]: 30% <- The Board.
Authority Given: Authority: Board Approval granted for this budget fraction <- The Board.
An example of allocating a budget. Notice the conditions "[From Next Year], [At least 10 signed], [1 in each country], [If Authority Given]." We could call this a 'conditional budget.'

6.3 Language Core: Resources, Budgets and Costs

Resource Requirement Specification

Resource requirements (Budgets) are specified in a similar manner to performance requirements, because they are *also* scalar requirements (that is, they are variable along a defined scale of measure). See Section 4.3, 'Language Core: Scalar Attributes.'

EXAMPLE Logic Space:
Type: Resource Requirement.
Scale: Maximum Storage Space in megabytes.
Owner: System Architecture.
Stakeholders: {Architect, Hardware Storage Designer, Handset User}.
Fail [Any One Function]: 100 Mb. "A resource constraint".
Budget [Any One <Frequent> Function]: 50 Mb. "A resource target".

6.4 Rules: Resource Requirement Specification

The rules for scalar requirement specification (Rules.SR) apply (*see Section 4.4*).

6.5 Process Description: Resource Requirement Specification

Process: Resource Requirement Specification

Tag: Process.RR.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: A process for specifying resource requirements and for cost estimating, resource budgeting, and project adjustment to stay within budgets.

Note: This process is highly iterative, and needs to be done early and often. It should actually be embedded in the Evolutionary result cycles. It is described here so that the reader sees the multiple elements of determining budgets. This is certainly not a simple procedure within a real project. Budgets will probably need to be estimated and adjusted several times, in the course of attempting to achieve a balance of the performance and function requirements with the resources.

Entry Conditions

E1: The Generic Entry Conditions apply. The specific source documents that should have already exited successfully from Specification Quality Control (SQC) include:

- the current requirements
- the design specifications
- any Impact Estimation tables (giving cost estimates for designs, or for Evo steps).

Note: If any of the source documents has failed to successfully exit SQC, then you can 'stipulate' desired costs, but you do not have a reasonable basis to confidently 'estimate' the costs of the designs/plans, which are needed to deliver the desired functionality and performance levels, on time.

Procedure

P1: **Identify Resource List:** Get existing lists of 'critical resources to be controlled' for this sort of project. These lists should include such things as project elapse times (long and short term), people, people work hours, project space, investments, development costs, production costs and operational costs (including maintenance costs).

P2: **Analyze Benchmark Costs:** Examine earlier similar projects for cost levels, and cost deviations from plans.

P3: **Determine Project Costs:** Determine acceptable and unacceptable cost levels for this project. *Consult any contracts, marketing plans and product plans.*

P4: **Produce Initial Project Budgets:** Specify an initial draft of project resource budgets.

P5: **Perform SQC:** Perform Specification Quality Control (SQC) using Rules.GS, Rules.RS, Rules.SD and Rules.SR. The source documents (process inputs) are listed in the entry condition E1 above. If the specification is not 'clean enough' (the SQC process calculates that there are one or more remaining major defects/page), then return to P1 and cycle through the procedure again as required.

P6: **Carry Out Evo:** Perform an Evo step. (Deliver some results!) Measure *real* costs, for the delivery, versus the budgeted step costs. Re-plan either costs or other things (such as designs, performance levels, and timing) in order to keep within the project resource requirements. Continue 'cycling' with this step until all the planned Evo steps for the project are completed.

Exit Conditions

X1: The Generic Exit Conditions apply.

The entire process of cost adjustment, and learning, goes on as long as money is still being spent on the project, or spent on the operational system. *'Project end' allows exit from the project. 'System/Product end' (that is, the system or product is no longer sold or distributed) allows exit from the system/product support process. This is a formal way of saying 'this is a continuing process, as long as resource is being consumed.'*

6.6 Principles: Resource Requirements

1. **The Principle of 'Many Critical Risks'**
   There are *many* resource, performance and condition dimensions critical to any system, not just one or a few.
2. **The Principle of 'You Can't Have It All, Trade-offs are a Necessity'**
   Fixing the required level of *one* resource dimension arbitrarily can only be done at the probable expense of *other* attributes.
3. **The Principle of 'You Get What You Pay For'**
   It is really the availability of resources, which limits the levels of performance that can be delivered in practice.

4. **The Principle of 'Attribute Balance'**
   Once you have found a balance between performance and costs, management cannot cut the financial budget, people or time without negative consequences.
5. **The Principle of 'The Cost of Perfection'**
   Perfect quality costs *infinity.*
6. **The Principle of 'The Rolls Royce'**
   Near-perfect performance levels cost more than most people would pay.
7. **The Principle of 'Natural Ambition'**
   The pressure on resources will *always* be at a 'level of discomfort', not to say downright intolerable – this is a natural management strategy to find out how far they can push!
8. **The Principle of 'The Traffic Bottleneck Illusion'**
   Increasing your allocated resources will *not* relieve the pressure on you, but only raise that sponsor's expectations.
   Removing one bottleneck serves mainly to discover others.
9. **The Principle of 'Really Useful Resource Management'**
   The only *practical* way to control costs and performance in large complex dynamic systems is by *early*, *frequent* realistic evolutionary *feedback* on costs, and consequent *adaptation* to realities.
10. **The Principle of 'Shifting Conflicts'**
    Conflicts amongst budget targets, performance targets and design ideas are natural; there's no blame. You just keep resolving them: it's the name of the game.
    Budget constraints will always exist and, will always be subject to change.

6.7 Additional Ideas

Using Impact Estimation and Evolutionary Processes to Balance Requirements

There are two Planguage methods that are worth outlining[1] at this point, because they are fundamental to the control of costs (and also performance).

One is Impact Estimation (IE), which enables *design evaluation* against *multiple* resource budgets and *multiple* performance targets. It produces

[1] See Chapter 9, 'Impact Estimation' and Chapter 10, 'Evolutionary Project Management' for more detail.

Table 6.1 A simple IE table.

Designs-> *Requirements*	*Contract*	*Supplier*	*Motive*	*Architect*	*Parts Used*	*Sum % Impacts*
Quality 1	0%	100%	50%	30%	−20%	160%
Quality 2	100%	50%	0%	20%	50%	220%
$Investment Cost	5%	10%	1%	10%	110%	136%
$Operational Cost	5%	50%	20%	1%	10%	86%
Staff Resource	10%	20%	10%	5%	0%	45%
Performance to Cost ratio	*100*/20	*150*/80	*50*/31	*50*/16	*30*/120	

an IE table that provides, amongst other information, performance to cost ratios, which allow relative assessment of the proposed designs.

The other method is Evolutionary Project Management (Evo), which plans and implements design delivery in a sequence of Evo steps. The choice of design for the next Evo step is re-evaluated once the feedback from the implementation of the latest Evo step is received. Evo can use the IE table information to select the design(s) for the next Evo step and to capture the feedback from past Evo steps.

The key point is that these two methods can evaluate *realistic feedback* from *partial implementation* of our designs. We get a *more reliable* picture of the *real* costs of what we are doing, and can then *make adjustments* to *anything* necessary (design, resources, performance levels and/or timing) to achieve the performance-to-cost ratio we are satisfied with.

Table 6.1 shows an example of a simple IE table.

This IE table has three *resource requirements*: $Investment Cost, $Operational Cost and Staff Resource. These are *defined* somewhere else, with a Past (Benchmark) level, which is represented by the 0% level on this table, and a Budget (or other Target) level, which is expressed by the 100% level on this table.

The *referenced* designs (Contract, Supplier, Motive, Architect, Parts Used) are also defined somewhere else with enough detail to permit us to estimate their impact, sufficiently well for our current purposes, on the performance goals (Quality 1 and Quality 2). Interpretations of impact are as follows:

- 0% is *no* change from the benchmark
- 100% reaches the target level on time
- 50% is *halfway* to the target level
- −20% is a '*negative*' impact compared to the benchmark.[2]

[2] For costs this would imply that a design earned resource rather than consumed it. This is not unthinkable.

We should arrive at the estimates of impact based on evidence (such as experiences with the defined design ideas).

Once each design idea has a numeric impact estimate for each performance cell and each cost cell, we can use these cell estimates to calculate a 'performance to cost' ratio. This is the overall ratio of performance delivered with respect to our objectives by the design idea (the sum of Quality 1 and Quality 2), over the sum of the estimated use of resources in relation to the plan by the design idea (the sum of the costs: {$Investment Cost, $Operational Cost, Staff Resource}).

Using a basic IE table, the impact of any design idea on performance with respect to its estimated costs can be evaluated. Design decisions, such as "what happens if we *drop* the design idea, Parts Used?" can also be assessed. Of course, the IE table simplifies, as all models do, but it still gives useful insights.

When there is a sufficient set of design ideas, that is likely to meet the planned levels, on time, with reasonable 'safety factors' (for example, all the 'Sum for Requirement' values are in excess of, say, 200%), then Evo can start to use the IE information in a slightly modified IE Table format to plan the implementation steps of the project.

In simple terms, an Evo plan would sequence the implementation of the design ideas to get the best results (the highest performance-to-cost ratios) delivered to stakeholders early. An IE table can be used after each evolutionary step delivery to capture the numeric feedback from the implementation of any set, or sub-set, of the designs, for any target market of interest *(see Chapter 10, 'Evolutionary Project Management,' for more detail).*

Instead of relying solely on estimates, *real* performance and cost experience is captured step by step and, of course, it can then be compared against the estimates step by step. This feedback on *real* cost and *real* performance levels allows better understanding of the true *future* cost levels, at an early stage of the project. This leads to better control over costs, system performance, design and projects. The heart of good project management is such multidimensional, numeric feedback and consequent improvement in plans.

6.8 Real Example: Resource Target and Resource Constraint Specification

Here is an example of a resource requirement specification, which includes some resource constraints. It also includes price specification. It is based on a real case study, but edited for confidentiality and to reflect the latest Planguage terminology.

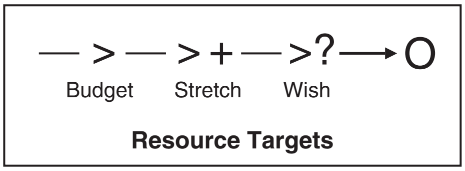
EXAMPLE **Installation Time:**
Ambition: Installation time must not be more than that of an unlicensed system <- RSW 3.
Type: Resource Requirement.
Installation Effort: Scale: Work Hours.
Budget [USA]: 15 <- Requirement Specification, Feb 5.
Installation Duration: Scale: Calendar Days.
Budget [USA]: 2.5 <- Requirement Specification, Feb 5.
Installation Costs:
Scale: Total Installation Cost of all Involved Parties.
Type: Resource Requirement.
Total Installation Cost: Defined As: Financial Cost of {Education of Customer People, Involvement during Installation of Customer People, Involvement during Planning of Customer People, Loss of Service in a PBX, Special Tools for Strange Cabling, any other thing even if not on this list!}
Past [DECT, USA, Last Year]: <not known exactly>.
Fail [Per Installation, USA, Release 1]: Maximum of twice DECT Installation Costs. "A constraint."
Budget [Per Installation, USA, Release 1]: Within ±20% of DECT Installation Costs. "A Target."
Per User Price:
Note: The actual price targets may vary from time to time and market to market.
Type: Performance Constraint.
Note: this is NOT a budget for the project or the Base Station system. This is a result of the design of the new system.
Scale: $ Per User Price for defined [Number of System Users] to use at a defined [Location] for defined [Release] of total Base Station {CE and RH}.
Past [Last Model]: $1,000.
Fail [30 to 250 System Users, USA, Release 1]: $700 or more <- RSW 2.
Survival [More than 250 System Users Or Larger Building Or tougher than Normal Radio, USA, Release 1]: $700 or more. <- RSW 2.
Subscriber Cost:
Type: Performance Constraint.
Note: The actual customer cost targets may vary from time to time and market to market.
Scale: $ Cost for a defined [Number of Users] of System per Subscriber, including TK and SW licenses cost to TeleCo.
Past: $600.
Fail [100 Users, USA, Release 1]: $400 or more <- RSW 2, Cost Assumptions (Page 2).
This is a real example, but not in its final form: it is only the first draft translation of a customer's older, non-Planguage specification. It is also upgraded with recent Planguage changes.

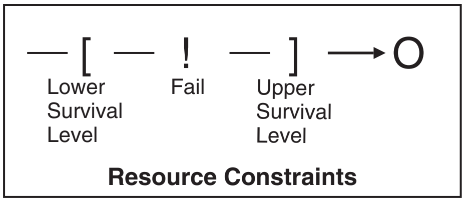
6.9 Diagrams/Icons: Resource Requirement Specification

Resource Requirement Icons

Resource target and constraint icons are scalar icons, identical to those used for performance attributes.

Resource targets specify how much we would 'like' to use of a resource. There are three types of resource target: Budget (>), Stretch (>+) and Wish (>?).





A Resource constraint is defined using a Fail concept (!) or a Survival concept: the '[' is a lower limit and the ']' is an upper limit.

Resource constraints set (relatively) strong framework limits to the use of resources. These strong constraints could be due to legal restrictions, contract limits or other sources, which are relatively inflexible. They are generally outside our control. Constraints are not so easily the subject of tradeoff decisions, as targets might well be.

Resource Requirement Specification Template

The scalar requirement template given in Section 4.9 should be used for resource requirement specification.

6.10 Summary: Resource Requirement Specification

There are many limited resources we must track for building, modifying and operating a system. Budget specifications will include calendar time, people effort, and money to implement, operate and service the system.

'Costs' is our term for 'use of resources': resources that are generally in demand for satisfying *other* priorities. Failure to think and document clearly with regard to resources is likely to lead to resource scarcity problems.

We assume that most systems that the reader is likely to use the Planguage methods on are non-trivial and difficult to manage. They are of such a nature that they are very difficult to *predict* costs for, and almost as difficult to *control* the costs of. Planguage addresses these problems in several ways:

- Planguage ensures specification of resource requirements is performed in a disciplined and detailed *numeric* way.
- Through Impact Estimation (IE), Planguage obtains *tightly integrated performance and cost information.* Not just the total final budgets, but *detailed budget allocation* at design idea level and at evolutionary step level, which is linked to the evolution of the stakeholder valued results! Such resource requirement specification information gives a better ability to predict costs in advance. Such resource budgeting is also important to ensure engineers do 'Design to Cost' from the earliest stages. It helps them keep aware that they do have finite limits for resources. It is otherwise too easy for them to focus on performance and technology; leaving serious cost considerations until too late.
- Through Evolutionary Project Management, Planguage provides better cost-expenditure control, because we have a way of *adjusting cost budgets and estimates for resource usage, as we learn, early and frequently, from practical experience.* Alternatively we can get resource control, because we can choose '*tradeoffs*' in order to maintain the budgets we *initially* planned for. 'Tradeoff' means that we can adjust certain performance levels and/or adjust certain design specifications. We can also adjust certain qualifiers [when, where, if]. With Planguage, we can more clearly, and earlier, see the exact options available, and make more intelligent tradeoff decisions.

The fundamental assumption of the Planguage method is that we must set things up to learn (this is Shewhart's Plan-Do-Study-Act cycle) as rapidly as possible, before we fail, and before our competitors do things better and 'put us out of business'. The threat of losing your workplace and budget to 'competition' applies even if you are a government agency or a charity!

By use of Planguage practices, the all-too-common project syndromes of 'running out of resource (time or money) without delivering any value' and 'pushing the system out of the door on the deadline; system performance be damned' ought to be eliminated for good! This is more than an optimistic hope. It has been done.

The real price of everything, what everything really costs to the man who wants to acquire it, is the toil and trouble of acquiring it.

Adam Smith (1723–90) Scottish economist,
The Wealth of Nations *(1776)*

Overview of Planguage Methods for Controlling Costs

The prerequisites for effective control over a project are tight integration of cost and performance considerations, 'design to cost' and using feedback on actual costs to modify plans. Planguage methods ensure these prerequisites by demanding:

- detailed, numeric, measurable performance specifications that adequately capture the performance requirements: the qualities (stakeholder-related objectives) as well as the workload capacities and resource savings (the resource-related objectives)
- resource requirement specifications for the resources allocated, and for any known restrictions on resource expenditure
- design specifications with detailed expected cost *and* performance attributes of the design
- impact estimates of the abilities of the various designs to meet both the performance goals and the resource budgets
- selection of evolutionary steps according to their stakeholder value, and their performance to cost ratios
- feedback from live systems of the actual progress towards achieving the performance levels, and the actual resource expenditure after implementing each evolutionary step
- action being taken on the feedback to adjust specifications, or the future evolutionary steps, to ensure realistic plans (revision of budgets or tradeoffs).

A Proposed Resource Requirement Specification Policy

1. **Define Resource Requirements Thoroughly**: In requirement specifications, all potentially critical resources shall be specified as budgets in a *well-defined, thorough* manner.
2. **Specify the Performance and Cost Relationship**: The level of both resource budget and performance goal detail shall be sufficient to enable us to understand the benefit, in relation to resources, of *incremental performance improvements*.
3. **Make All Cost Requirements Visible**: We must be able to 'see' all opportunities to reduce costs by investment in better system design. The budgets must specifically incorporate ongoing *operational costs* requirements (that is, the resources required for such things as installation, adaptation, porting, maintenance, recovery, auditing, servicing and/or customer help lines) so these can *compete for priority* with *short-term investment* costs.

4. **Plan for the Long Term**: All budgets shall consider the *total* lifetime of system perspectives. This specifically includes *long-term* considerations (such as costs of system retirement, pollution and accidents).
5. **Designs Shall Be Cost Estimated for Impact on All Critical Resources**: Costs shall be estimated for all critical and budgeted multiple resource factors for *every discrete design idea* using Impact Estimation Tables.
6. **Let Value Decide the Costs**: Value delivered in relation to costs, not 'resources consumed' alone, should dictate expenditure. If designs provide the opportunity for excellent required 'payback', then we should automatically spend more, and vice versa. *(Given that budgets are formulated in 'performance to cost' terms, and we have Evolutionary feedback, the levels of risk should be under acceptable control.)*
7. **Document Supporting Information**: When defining cost requirements, *full documentation* shall be given about assumptions, benchmarks, risks, uncertainties, ranges, authorities, sources and other related facts so as to give us the best possible background for rapid, confident, independent decision-making by the systems engineers and managers.
8. **Justify Estimates and Perform Specification Quality Control**: When making estimates, the full array of *evidence and sources* of the evidence shall be documented. Worst-case scenarios shall be given explicitly. The estimations shall undergo *Specification Quality Control* (SQC).
9. **Track Costs Early, During Implementation**: Costs shall be tracked and analyzed *at every evolutionary step* of development, so as to learn of problems as early as possible, and take corrective action.

This policy above captures many of the key points discussed in this chapter about Resource Requirements. Note: this policy should also be supported by specification rules to enable the Specification Quality Control (SQC) of Resource Requirements and Cost Estimates.

Chapter

7

Design Ideas and Design Engineering

How to Solve the 'Requirements Problem'

Glossary Concepts

Design Idea

Design Specification

Design Engineering

7.1 Introduction

To Design and to Engineer

The basic design process is finding 'means' for 'ends': it is finding designs that match the requirements.

What is the difference between design and design engineering? They are both essentially the same generic, and basic, process of 'finding satisfactory designs.' However, engineering disciplines are characterized, in my opinion, by the following distinctive traits:

- *quantification* of variable ideas (not 'high', but '42')
- concern for *all* necessary factors (all stakeholders, all requirements and all known design options)
- concern for more than mere 'satisfaction'; concern for *competitive* optimization – 'being the best', rather than just 'getting along'
- rational and systematic argument (for example, the use of Impact Estimation tables to discuss or present design quantitatively with respect to facts, not 'less formal' design or 'emotional' design.

Design asks, ''Is this a good design?''
Design Engineering asks, ''What are the totality of performance and cost attributes expected from *this design* in relation to the multiple, quantified, performance and cost *requirements*? What are the risks, priorities, uncertainties, issues, relationships, dependencies and long-term lifecycle considerations, that we should responsibly consider about this design?''

Requirement Specification, Design Engineering and Evo are all Iterative Processes

Design ideas emerge, and are refined, throughout the lifetime of a system. Iteration is necessary in order to improve both the design ideas and the related requirements. Requirements and design ideas *cannot* be determined well in one single pass. Feedback from initial *design engineering* processes is necessary to get a *realistic* idea of which *design ideas are possible*, to determine how much design ideas *might cost* and to identify which tradeoffs amongst performance levels *might have to* be made. Until *the design and the requirements* are adjusted to this 'balanced level' with regard to reality, it is not possible to 'finalize' a competitive *design* for implementation.

In addition, after implementation starts, as a result of the measurable *feedback* obtained from the delivery of each of the Evo steps, even

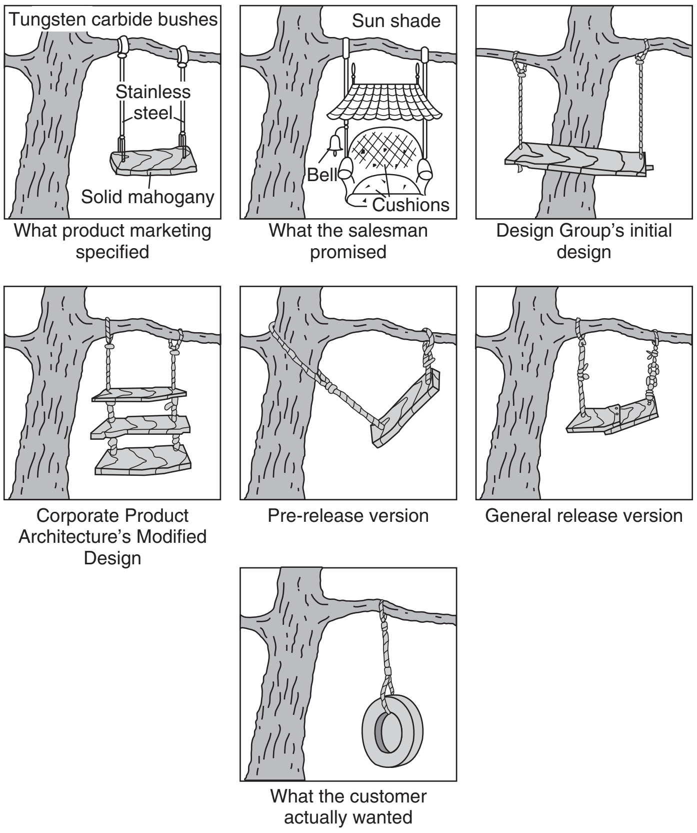

Tungsten carbide bushes
Stainless steel
Solid mahogany
What product marketing specified
Sun shade
Bell
Cushions
What the salesman promised
Design Group's initial design
Corporate Product Architecture's Modified Design
Pre-release version
General release version
What the customer actually wanted

Figure 7.1
The swing solutions. Source: Anon.

further refinement has to be considered for the design ideas, the requirements and the *implementation plans*.

> *Requirement Specification, Design Engineering and Evo are intimately linked, and some iteration linking them is necessary to get the best competitive results.*

Requirements Dictate and Constrain the Design, but *Detailed* Requirement Specification can Wait

What stakeholders perceive as 'value' drives us to state 'what stakeholders want' and 'how much stakeholders might be willing to pay for such change': in other words, to state the *requirements*. Requirements, which reflect values, give us a sound basis for evaluating a design idea: a basis for deciding if we might get what we will find of 'value' from a potential design idea.

'Interesting' results are our 'values.'

Keeney (1992)

We have to have at least a preliminary set of requirements, before we are ready to '*design*.'[1] These requirements could, even for a large project, be as simple as a statement of the handful of most critical requirements. (After all, these critical requirements are in fact usually driving the investment and the project!) The more detailed requirements can be derived gradually, as needed, during the Evolutionary Project Management (Evo) process. There is no need to try to get all the detail immediately. In fact, there is some virtue in letting the detail emerge as a function of experience and of interaction with key stakeholders.

Note: The detail is, however, ultimately important, and must be eventually specified, so we can fully understand the meaning, intent, risks, assumptions and dependencies of all the requirements. For example, we need to understand which requirements are targeted only at specific system components.

Any Design Idea[2] can be Considered

Any design idea that potentially contributes to the solution of the requirements, can be *suggested*. It is a question of how much a design idea contributes towards meeting the requirements, and at what costs, which determines whether a design is finally selected and implemented. It is then the design idea's *real* performance, on delivery, that will determine whether it survives, or must be replaced by another design idea.

EXAMPLE Some Design Ideas:

- using process improvement teams
- allowing the project team an extra day's time off if a deadline is successfully met (motivation)
- buying an extra server (buy hardware)
- giving discounts to customers who field trial new products (monetary motivation)
- buying a standard component (buy hardware with known characteristics)
- contracting for a special tailored component (subcontracting and tailoring)
- building our own software component (development in-house)
- improving testing process (improving a specific development process).

This example shows a wide variation of types of design.

[1] In this book generally, I use the term 'to design,' but with regard Planguage processes, I actually mean 'to design *engineer*,' that is, to use rational and quantitative approaches.
[2] The term 'design idea' is used in this chapter. Solution, idea, strategy, design, means, idea and design solution are all synonyms amongst many other synonyms for 'design idea.'

Design Ideas can be Identified during Requirement Specification

Even while you are initially specifying *requirements*, you should, if you feel that design ideas are flowing into your mind or the minds of colleagues, develop two separate lists of design ideas: *potential design ideas* and *design constraints*. The potential design ideas can then be kept aside for serious consideration in the *design* phase.

Potential Design Ideas

These are either design ideas that were first assumed to be *requirements*, but then were recognized as *really* being *optional* designs, or they are simply design ideas that surfaced during requirement specification. Sometimes such design ideas are deliberately 'brainstormed' (for example, if experts in a specific area are available only during the initial requirements' gathering, then capturing their design ideas might be opportune). Here is an example of a way of keeping track of any potential design ideas; there is *no commitment* to implementing them at *this* stage.

EXAMPLE Availability:
Type: Quality Requirement.
Scale: % Uptime.
Goal [USA, Version 1.0]: 99.90% <- Marketing Plan [April 20, This Year].
Design A [Availability = 99.90%]: Design Idea: Reuse of <high MTBF> Components <- Ed's suggestion.
Stretch [Worldwide, Version 3.0 and on]: 99.998% <- CEO Vision, "World Class."
Design B [Availability = 99.998%]: Design Idea: Triple Redundant Distinct Software <- Mike.
The two design idea specifications are local to the two different target specifications. They are not design constraints. They are clearly suggestions that need to be evaluated like any other suggestion.

Allowing systems engineers to note design ideas at an early stage is useful in several ways:

- it keeps track of potentially valuable design ideas which otherwise might get forgotten
- it helps make the distinction between the requirements and the design technology clearer ('clear ends–means separation')
- it lends credibility to the proposed goal levels (there exists some credible technology for the goal level suggested)

- it avoids the 'frustration' that some systems engineers feel when they are not allowed to be specific about the technology they have in mind
- it allows us to send a message that we have noted a systems engineer's suggestion or 'pet idea' and credited them with it – without yet officially approving it.

Some of the early design ideas may be *politically* wise to consider, due to the fact that influential stakeholders have suggested them. There is no risk of any unfairness in considering these design ideas, because they will have to compete with the later design ideas. All design ideas must win their place for implementation by being the best, in terms of numeric satisfaction of the requirements.

Design Constraints

These are design ideas within the requirement specification, which have to be implemented at some stage. They can either specify or veto the use of specific designs. Usually, specific qualifying conditions apply.

EXAMPLE Project Interface [Product Line = New Generation, European Market]:
Type: Design Constraint.
Description: The full Project Interface shall be implemented using the most <current version> available. It shall be updated whenever <newer versions> are available.
Rationale: Project Consortium Agreements.

This design constraint (a requirement) applies only to the Product Line of New Generation within the European Market.

The Need for Alternative Design Ideas

Choosing the Best from the Alternatives

When searching to find design ideas, it is important to look for alternative design ideas. Each individual design idea will produce different effects on a system's scalar attributes: the resource usage and performance levels. It is a question of selecting the design idea which has the best performance to cost ratio or the 'best fit to the requirements' with regard to 'delivering stakeholder value' compared to 'resources used' (value to cost ratio).

Choosing the Best Combined Set from all the Alternatives

Design ideas put together in different combinations will interact with each other in different ways: there could be negative side effects and/or positive 'combining' effects (synergy). By having several alternatives, it is possible to select the combination of design ideas, which has the best, estimated impact on the requirements. (Of course, the chosen combination can always be altered over time, in the light of feedback from evolutionary delivery.)

Reducing Risk by Use of Alternative Design

Another main reason for having alternatives is to reduce risk. If there are several candidate design ideas, then if the first choice fails there is always a backup. At an extreme, alternative design ideas may be implemented in parallel to ensure that specific requirements are fully met.

Design Optimization

When you are designing, you need to decide what type of optimization strategy you intend to use. The strategy options for Design Optimization Tradeoff include:

- **Cost Minimization**: When performance targets are met by specified designs, we can choose to continue to find alternative designs, that are at least equally well performing, with a view to reducing costs to the cost targets (if not below them!).
- **Design to Cost**: Another approach would be to design *to fully use* the all budgeted resources and to look for the designs that give maximum impact on the performance targets. In other words, the *most value* for a specific amount of limited resources. This is called 'Design to Cost.' '*Cutting your coat to suit your cloth.*'

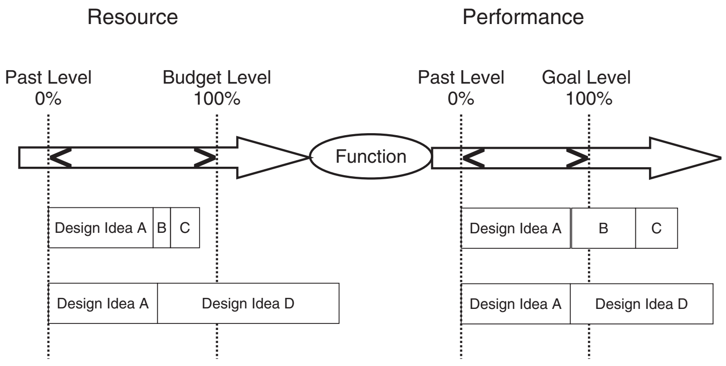


Figure 7.2
To 'design' is to find design ideas, like A, B, and C, which will contribute towards planned performance and resource levels, while simultaneously respecting all constraints. Design Idea D is 'good,' but costs too much.

- **Design to Performance Targets within Cost**: Another option would be to design to meet all the performance targets within cost, but to stop the process once all the planned performance levels were met. In other words, do not use additional time to reduce resource utilization further. This could be a possible approach when Time to Market is the most critical resource.
- **Design for Risk**: Another optimization concept would be to design with regard to risk. The *most pessimistic* estimates of performance impact, and of costs, would be used to determine the 'best design'. There are many other devices in Planguage that help us consider risk when designing *(see specifically, Impact Estimation).*

There are more combinations than those mentioned above. But you can see some basic choices. It is important in any project that you recognize how you are approaching the design optimization process, and that you communicate with your management about it. It could be there is some misunderstanding – maybe there *is* more financial budget available from them, as long as you show a track record of successful delivery.

If there are specific resource budgets that are critical to you, such as 'Time to Market', we recommend that you *initially* 'Design to Cost' with respect to 'calendar time' for delivery to market. Generally, you will want to design to meet the most critical constraints first, then see if you can maximize delivery of performance attributes and minimize other cost aspects in a second round of design effort.

Brief Recap of Planguage Methods and the Design Engineering Process

See Figures 1.3 and 1.6 and Table 1.1 in Chapter 1, 'Planguage Basics and Process Control,' and also the generic project process in Section 1.5. These show how the Design Engineering Process fits into the overall Planguage process model.

Specifically, with regards the Design Engineering Process:

- **Requirement Specification** supports the design engineering process by capturing the requirements. The requirements include specific information required for design decision-making. (For example, see further discussion on 'Priority Determination' in Section 7.7.)
- **Impact Estimation** (IE) is part of the Design Engineering Process. It is the Planguage method used to *evaluate and choose* design ideas. It also incorporates risk evaluation. In addition, IE can also be used to monitor the actual progress towards meeting the requirements. The actual step measurements, obtained after each Evo step has been completed (with delivery of one or more design ideas), can be

input into an IE table and compared against the original estimates. This feedback is used by the design engineering process, to understand where the gaps in design actually exist (that is, the gaps, which require additional design). (*See Chapter 9 for further details on IE.*)

- **Evolutionary Project Management** (**Evo**) is used to *actually deliver* the design ideas. Evo handles risk by several means:
    - implementing design, step by step
    - demanding that we choose the design ideas *most likely* to provide *high benefit* (highest value to cost ratios, highest performance to cost ratios) for *early* delivery (design ideas are 'sequenced' by some chosen evaluation of their potential benefits and costs into an Evo step plan)
    - testing the reality of the design ideas 'in the field'
    - providing and using feedback data after each step. We can then realistically understand the accuracy of our estimates, concerning design ideas, and can take appropriate measures, depending on the level of risk we perceive
    - we have incrementally 'banked' *some* results and eliminated *some* risk, which maybe means we can afford to discuss taking some higher risk steps.

Note: Both the requirement specification process and the design engineering process are incorporated into Evo; each result cycle demands re-evaluation of the design and brief re-evaluation of the requirements (possible adjustments and tradeoffs) (see Chapter 10). As stated earlier in this section, there is continuous iteration amongst these processes.

7.2 Practical Example: Beginning the Design Engineering Process

Let us say we have specified the following requirements for a project 'Staging a Conference':

Staging a Conference: Type = Function.
========== Conference Performance Requirements =========
Participation: Quality Requirement:
Scale: Percentage of Worldwide Membership participating.
Goal: 10%.
Representation:
Scale: Percentage of Worldwide Membership represented within defined <groups>.
Goal [Age under 25 or equating to <Student Status>]: 10%.
Information:
Scale: Percentage of Talks rated as 'good' or better (5+ on feedback sheet scale).
Goal: 50%.

Conviction:
Scale: Percentage of Participants wanting to return Next Conference.
Goal: 80%.
Influence:
Scale: Percentage of Participants who <improve as result of the Conference>.
Past: 90%.
Goal: 95%.
Fun:
Scale: Percentage of Participants rating the Conference City quality as 'good' or better (5+ on Feedback Sheet scale).
Past: 45%.
Goal: 60%.
============ Conference Budget Requirements ============
Financial Cost: Resource Requirement [Financial]:
Scale: Average Participant Conference Cost for an individual Participant including Travel Costs.
Fail: Less than $2,000.
Budget: Less than $1,200.
A set of requirements for a conference, mostly performance requirements and one budget.

Now we can, driven by these relatively clear requirements, start designing.

We begin by listing any design constraints – the 'given' *design* ideas. Here there are none, the only 'given' is the main function that we are to stage a conference.

We can then list 'at least one potential design idea, for *each* of the requirements.' This is an arbitrary way of covering the requirements with 'some' design.

Design Ideas:
Central: Choose a location in the membership center of gravity (New York?).
Youth: Suggest and support local campaigns to finance 'sending' a young representative to conference.
Facts: Review all submitted papers on <content>.
London: Announce that the conference is to be in London the next year.
Diploma: Give diplomas for attendance, and additional diplomas for individual tutorial courses.
Events: Have entertainment activities organized every evening, such as river tours.
Discounts: Get discounts on airfare and hotels.

Now, are these design ideas going to make the conference what we want it to be, as defined by the target levels? Nobody really knows and nobody can say. Why not? Well, it depends on the *interpretation* of the design ideas and

Table 7.1 Impact Estimation table

Design Ideas Requirements	*Central*	*Youth*	*Facts*	*London*	*Diploma*	*Events*	*Discounts*	*Sum for Requirement*
Performance Requirements								
Participation	80% ±50%	60% ±70%	0% ±50%	0% ±50%	30% ±50%	20% ±50%	30% ±50%	220% ±370%
Representation	80% ±50%	80% ± 50%	10% ±50%	0% ±50%	10% ±50%	20% ±50%	50% ±40%	250% ±340%
Information	0% ±50%	20% ±40%	80% ±50%	0% ±20%	20% ±50%	0% ±50%	0% ±30%	120% ±290%
Conviction	0% ±10%	20% ±50%	60% ±30%	80% ±50%	10% ±50%	80% ±50%	0% ±50%	250% ±290%
Influence	0% ±50%	40% ±40%	60% ±50%	0% ±50%	80% ±50%	80% ±5%	0% ±50%	260% ±340%
Fun	50% ±50%	40% ±50%	10% ±50%	0% ±0%	0% ±0%	80% ±50%	0% ±0%	180% ±200%
Sum of Performance	210% ±260%	260% ±300%	220% ±280%	80% ±220%	150% ±250%	280% ±300%	80% ±220%	
Resource Requirements								
Financial Cost	20% ±30%	1% ±1%	1% ±1%	1% ±1%	1% ±5%	30% ±50%	30% ±50%	111% ±135%
Performance to Cost Ratio	210/20	260/1	220/1	80/1	150/1	280/30	80/30	

their *execution* in practice. Can we influence that? Yes. By specifying a more detailed design specification with precise details of what we are going to do, and exactly how it is to be done in practice (the implementation and operational design detail). In other words, we now have that first 'sketch of the building,' but we need to get down to the detailed 'blueprints' (engineering) needed by the 'bricklayers and carpenters.'

The first step is to assess what we can evaluate about the impact of our proposed design ideas on the requirements (perhaps a little exaggerated to make our point).

The Impact Estimation table is a way to 'see' what we are doing. A 100% estimate on this table is a belief (right or wrong, well founded or not) that we *will* reach the planned level on time. The plus/minus estimate is a *rough* notion of the uncertainty. Until we get better definition and justification, these numbers are of only slightly better value than words, such as good, bad, excellent. But they do give us *a systematic basis for improvement* in our planning.

(I ask the reader to be patient; a proper version of Impact Estimation is presented in Chapter 9. All I am doing here is illustrating how one might define some requirements and design ideas, and then evaluate the impacts of each of the design ideas on all the requirements.)

The first observation I would make *here* is that we need to *redefine* the design ideas, *with more detail.* This is because of the high plus/minus uncertainties specified.

EXAMPLE

Central: Town must be cheaply accessible by most Participants. Location itself must offer reasonable priced Accommodation (like university dorms) within walking distance of the Conference Facilities. Easy access to shops, restaurants, entertainment. <Add even more, and give concrete suggestions>

The ideal would be to create a hierarchy of the components of the design idea, Central and evaluate each separately, to home in on exactly what aspects of the design idea gave most stakeholder value.

7.3 Language Core: Design Idea Specification

Design specification is not just writing down the bare outline of the 'design idea' itself. You have the option of including a large number of additional parameters to describe the design idea. Why bother? Well, it is a matter of how much you want to force yourself to think about your idea, how much you want to share in writing with others, and how much you want to control any risks involved with the design. You must have reasonable confidence that the design idea really will deliver the results you have estimated.

In the design specification, you should ensure that you:

- Supply more detail in the Definition parameter, as this will lead to better understanding of its specific performance and cost impacts. This can be done using structural breakdown (see the Definition parameter in 'Transport by Buses' example below). Each of the sub-design ideas can be refined, until you feel that you have enough detail in the design ideas to guarantee the results levels and result timings, which are planned across all the requirements (or you identify that you need additional design ideas).
- Clarify and limit the design ideas to the specific ones that you want. Avoid ambiguity so that other people can't misinterpret your design intent.
- Identify and specify designs that clearly deliver at least partly one required performance attribute. Any 'side-effect' impacts of each design, on the other requirements, must also be analyzed and estimated. Use the '->' Impacts parameter to explicitly declare which attributes you hope, or expect, will be impacted by specific sub-design ideas. ("Design Idea A -> Safety.")

Of course, you need to tailor your design specifications to suit the circumstances. A simple rule to guide you is 'to try the design specification parameters out at least once.' Too much description of a design idea will not hurt you, and can easily be deleted if it does not serve a useful purpose. Observe what the engineering team feels is worthwhile, and use that level of specification.

Here is an example of specifying a design idea. It was actually used in charity relief-organization work. (It shows how the main idea can be supported by sub-designs. The aim being to get better control over the results.)

EXAMPLE Transport by Buses: Design Idea.
Description: Drive Refugees back across the border by bus.
Definition [Sub-designs]:
Village: Refugees should be selected from the same, or nearby, village -> Financial Cost.
White Paint: Buses should be painted UN white, and UN marked -> Safety <- Geneva Convention, Article 6.3.
Agreement: <Agreement with government> to allow transport and resettlement, without harassment, shall be made *before* crossing the border. Agreement papers will be onboard the bus -> Safety.
Radio: Buses shall have radio or mobile telephone contact with our headquarters *during* the transport -> Safety. "Maybe also video and tape recorder?"
Witness: UN employees, or relief agency employees, perhaps UN soldiers will accompany the buses -> Safety.

Example of a Design Specification

Tag: OPP Integration.
Type: Design Idea [Architectural].
================================ Basic Information ================================
Version:
Status:
Quality Level:
Owner:
Expert:
Authority:
Source: System Specification Volume 1 Version 1.1, SIG, February 4 – Precise reference <to be supplied by Andy>.

Gist: The X-999 would integrate both 'Push Server' and 'Push Client' roles of the Object Push Profile (OPP).
Description: Defined X-999 software acts in accordance with the <specification> defined for both the Push Server and Push Client roles of the Object Push Profile (OPP).
Only when official certification is actually and correctly granted; has the {developer or supplier or any real integrator, whoever it really is doing the integration} completed their task correctly.
This includes correct proven interface to any other related modules specified in the specification.

Stakeholders: Phonebook, Scheduler, Testers, <Product Architect>, Product Planner, Software Engineers, User Interface Designer, Project Team Leader, Company engineers, Developers from other Company product departments which we interface with, the supplier of the TTT, CC. "Other than Owner and Expert. The people we are writing this particular requirement for."

============================== Design Relationships ==============================
Reuse of Other Design:
Reuse of This Design:
Design Constraints:
Sub-Designs:

============================== Impacts Relationships ==============================
Impacts [Functions]:
Impacts [Intended]: Interoperability.
Impacts [Side Effects]:
Impacts [Costs]:
Impacts [Other Designs]:
Interoperability: Defined As: Certified that this device can exchange information with any other device produced by this project.

=========================== Impact Estimation/Feedback ===========================
Tag: Interoperability.
Scale:
Percentage Impact [Interoperability, Estimate]: <100% of Interoperability objective with other devices that support OPP on time is estimated to be the result>.
=========================== Priority and Risk Management ===========================
Rationale:
Value:
Assumptions: There are some performance requirements within our certification process regarding probability of connection and transmission etc. that we do not remember <-TG.
Dependencies:
Risks:
We do not 'understand' fully (because we don't have information to hand here) our certification requirements, so we risk that our design will fail certification <-TG.
Priority:
Issues:
============================= Implementation Control =============================
Not yet filled in.
============================= Location of Specification =============================
Location of Master Specification: <Give the intranet web location of this master specification>.

Figure 7.3
Here is a real (doctored!) example of a design specification using a version of the Design Specification Template given later in Section 7.9. Not all parameters are filled out yet. Notice that even the parameters which are not filled out (like Impacts [Side effects] and Issues) are asking important questions about the design – and hinting that responsible designers should answer such questions!

7.4 Rules: Design Specification

Tag: Rules.DS.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Note: Design specifications are either for optional design ideas (possible solutions) or required design constraints (that is, actual requirements AND consequently, pre-selected solutions).

Base: The rules for generic specification, Rules.GS apply. If the design idea is a design constraint (a requirement), the rules for requirement specification, Rules.RS also apply.

R1: **Design Separation**: Only design ideas that are intentionally 'constraints' (*Type: Design Constraint*) are specified in the *requirements.* Any other design ideas are specified separately (*Type: Design Idea*). *Note all the design ideas specified as requirements should be explicitly identified as 'Design Constraints.' (Repeat of Rules.RS.R9: Design Separation.)*

R2: **Detail**: A design specification should be specified in *enough detail* so that we know precisely what is expected, and do not, and cannot, inadvertently assume or include design elements, which are not actually intended. It should be 'foolproof.' *For complex designs, the detailed definition of its sub-designs can satisfy this need for clarity, the highest level design description does not need to hold all the detail.*

R3: **Explode**: Any design idea (Type: Complex Design Idea), whose impact on attributes can be better controlled by detailing it, should be broken down into a list of the tag names of its elementary and/or complex sub-design ideas. *Use the parameter 'Definition' for Sub-Designs.*

If you know it can be decomposed; but don't want to decompose it just now, at least explicitly indicate the potential of such a breakdown. *Use a Comment or Note parameter.*

R4: **Dependencies**: Any known dependencies for successful implementation of a design idea need to be specified explicitly. Nothing should be assumed to be 'obvious.' *Use the parameter, Dependency (or Depends On), or other suitable notation such as [qualifiers].*

(For design constraints (requirements), this is a repeat of the rule, Rules.RS.R5: Dependencies.)

R5: **Impacts:** For each design idea, specify *at least one* main performance attribute impacted by it. *Use an impact arrow '->' or the Impacts parameter.*

Comment: At early stages of design specification, you are just establishing that the design idea has some relevance to meeting your requirements. Later, an IE table can be used to establish the performance to cost ratio and/or the value to cost ratio of each design idea.

EXAMPLE Design Idea 1 -> Availability.
Design Tag 2: Design Idea.
Impacts: Performance X.

R6: **Side Effects**: Document in the design specification any side effects of the design idea (on defined requirements or other specified potential design ideas) that you expect or fear. *Do this using explicit parameters, such as Risks, Impacts [Side Effect] and Assumptions.*

Do not assume others will know, suspect or bother to deal with risks, side effects and assumptions. Do it yourself. Understanding potential side effects is a sign of your system engineering competence and maturity. Don't be shy!

EXAMPLE Design Idea 5: Have a <circus> -> Cost A.
Risk [Design Idea 5]: This might cost us more than justified.
Design Idea 6: Hold the conference in Acapulco.
Risk: Students might not be able to afford attendance at such a place?
Design Idea 7: Use Widget Model 2.3.
Assumption: Cost of purchasing quantities of 100 or more is 40% less due to discount.
Impacts [Side Effects]: {Reliability, Usability}.

R7: **Background Information**: Capture the background information for any estimated or actual *impact* of a design idea on a performance/cost attribute. The evidence supporting the impact, the level of uncertainty (the error margins), the level of credibility of any information and the source(s) for all this information should be given as far as possible. For example, state a previous project's experience of using the design idea. *Use Evidence, Uncertainty, Credibility, and Source parameters.*

Comment: This helps 'ground' opinions on how the design ideas contribute to meeting the requirements. It is also preparation for filling out an IE table.

EXAMPLE Design Tag 2 -> Performance X <- Source Y.

R8: **IE table**: The set of design ideas specified to meet a set of requirements should be validated at an early stage by using an Impact Estimation (IE) table.

Does the selected set of design ideas produce a good enough set of expected attributes, with respect to all requirements and any other proposed design *ideas? Use an IE table as a working tool when specifying design ideas and also, when performing quality control or design reviews on design idea specifications.*

See Chapter 9, 'Impact Estimation.' Failing that, at least ask the 'Twelve Tough Questions' about the design ideas! (Can you quantify the impacts?) See Section 1.2 for details of the 'Twelve Tough Questions.'

R9: **Constraints**: No single design specification, or set of design specifications cumulatively, can violate any specified constraint. If there is *any* risk that this might occur, the system engineer will give a suitable warning signal. *Use the Risk or Issues parameters, for example.*

R10: **Rejected Designs**: A design idea may be declared 'rejected' for any number of reasons. It should be retained in the design documentation or database, with information showing that it was rejected, and also, why it was rejected and by whom.

EXAMPLE

Design Idea D: Design Idea.
Status: Rejected.
Rationale [Status]: Exceeds Operational Costs.
Authority: Mary Fine. Date [Status]: April 20, This Year.

7.5 Process Description: The Design Engineering Process

Process: Design Engineering Process

Tag: Process.DE.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Assumption: We have clearly-stated and reasonably complete requirements.

Notes:

1. Design is an iterative process. The process given in this section should be viewed with this in mind; the procedure is written as if it were carried out in a single pass, but in practice, a much more complex pattern of cross-checking, backtracking and tradeoffs would actually be carried out.
2. This procedure is much longer than it needs to be, due to the nature of this book. You should probably use a more concise version (say, one statement for each procedure step).

Overview of a Design Process

Design is an *intellectual* process, which is supported by problem definition, requirement specification, best-practice design process standards and analysis tools. The following fundamental questions arise in designing:

1. **Analyze the Requirements:** Which requirements are of high stakeholder value? What constraints apply? What is the priority time sequencing for delivery of requirements?
2. **Find and Specify Design Ideas:** How do we *find* and *specify* potential design solutions for our requirements?
3. **Evaluate Design Ideas:** How do we *evaluate* potential design solutions?
4. **Select Design Ideas and Produce Evo Plan:** How do we *choose* from several 'good' design alternatives? What do we do about *uncertainties*, and about the *risk* that the selected designs are not as good as we thought?

Notes:

Ansoff points out that "Simon has shown that solution of any decision problem in business, science, or art can be viewed in four steps:

1. *PERCEPTION of decision need or opportunity. Simon calls this the INTELLIGENCE phase.*
2. *FORMULATION of alternative courses of action.*
3. *EVALUATION of the alternatives for their respective contributions.*
4. *CHOICE of one or more alternatives for implementation." (Ansoff 1965)*

This supports the choice of the four main sub-processes of the Design Process!

Figure 7.4
Overview of a Design Process. This applies with or without a quantified engineering approach to design.

Entry Conditions

E1: The Generic Entry Conditions apply. The requirement specification should ideally have exited from Specification Quality Control (SQC).

E2: Any existing feedback, from Impact Estimation (design idea analysis), or practical trials, is made available to the design engineer.

Procedure

P1: **Analyze the Requirements**: You may well identify stakeholder conflicts and overlaps[3] amongst the requirements while analyzing them. These need conflict resolution: consider if the 'tougher' requirement level can be used, identify the 'owning' stakeholders for all values and negotiate with the stakeholders. It may well be worth waiting until you have some alternative design ideas before you negotiate with the stakeholders, as by then you will have a better understanding of

[3] Overlaps in requirements represent either an opportunity for additional value to be delivered or, the possibility for over-estimation of value *('double accounting')*.

what solutions can be delivered. See also discussion in Section 7.7 on Priority Management.

P1.1: **Establish the Stakeholder Value on Delivery of each Requirement**: The value on delivery of each requirement to the system/organization should be assessed. You are looking for the requirement areas where there are major benefits. (What is of value depends on the stakeholders: it might not be just financial resource.) Identify the volume of use associated with each requirement.

Ideally, the stakeholders will have already selected the highest value requirements as the *critical requirements*.

For example: resource savings for performance requirements will be relatively simple to determine. Say you wanted to bring the time of carrying out a transaction down from two minutes to one minute. The benefit to the business, assuming 200 such transactions were carried out a day, would be 200 minutes per day. If operators had to wait while each transaction went through – this could amount to freeing up over three work-hours each day to carry out additional activities. In other words, assuming 250 work-days each year, 250 multiplied by 200 minutes each year. To the business, the ability to free up staff or the cost of employing the staff in this area, is the 'value' gained on delivering the requirement.

P1.2: **Sequence the Delivery Order of the Requirements**: Sequence the requirements for attention in the order of maximum stakeholder value first. Adjust to cope with any dependencies amongst the implementation of requirements (These dependencies either prevent implementation or prevent some level of benefit being achieved).

Note also the possibility for delivering each requirement in stages either by gradually improving a performance level or cost level or by delivering into different areas at different times (that is, to divide according to qualifier conditions; for example, by geographic area, by role and/or by timescale).

P1.3: **Establish Scope of Design Interest**: Establish and specify the scope of interest for the system design. This will be a set of qualifying conditions covering a specific timeframe and specific system space (locations/components/functionality). For larger systems, you might want to divide the system into major subsets – maybe, say, by functionality and/or by timescale, so that you can work on a series of smaller design areas.

P1.4: **Make a List of Requirements within the Scope defined**: Identify any function, performance, resource or condition *constraints* specified in the requirements. Then identify the *target* requirements. Note the qualifying conditions, which apply to each one of them.

P2: **Find and Specify Design Ideas**:

P2.1: **Exploit any Earlier Notes of Design Ideas**: Check to see if there is already a list of potential design ideas developed at the same time as the requirement specification. If there is, include those design ideas for consideration.

P2.2: **Establish the Design Constraints**: Read the requirements to see if there are any design constraints specified (Type: Design Constraint). If there are, then note them and any specific conditions qualifying them [time, place, event].

P2.3: **Brainstorm Design Ideas**: Search all available sources of design information for good matches to our stated requirements. (Specifically with regard to meeting the function requirements, achieving the performance levels and, delivering within the budgets. Any constraints and conditions must also be considered.)

Identify any dependencies amongst design ideas.

Also identify any design ideas that are alternatives.

Attention needs to be focused especially on the areas of greatest benefit to the system/organization. It is *the gaps* between our current updated system design process benchmarks (how well we have satisfied requirements until now) and our specified targets, which are of interest.

We are totally dependent in our search on the following:

- Knowledge of the *existence* of good design ideas (Where are they? Do we have the best ones?).
- Having complete and reliable *information about the likely impacts* of the design ideas on system attributes, so we can match the best design ideas to our residual requirement gaps. Most design ideas have too little specified, or available, data about their performance and cost characteristics.
- Understanding how design ideas *mix and interact* with each other. (Maybe the mixture will conflict? A design idea, in itself, might seem satisfactory, but the effect of combination with other design ideas, already in place, or under consideration, could be a counter-productive. Alcohol and driving don't mix well; though each in the right time and place might be acceptable.)

Hint: Select design ideas from available knowledge: books, periodicals, conference proceedings, past products, memory, colleagues, web searches, company experience, competitive analysis, benchmarking and others.

Stop the search when a set of satisfactory design ideas has been found, or when you run out of time to search for more.

P2.4: **Draft Design Ideas**: Draft a set of the design ideas, which might satisfy the requirements. See the design specification template outlined in Section 7.9.

P2.5: **Collect Specification Detail (to support later Impact Estimation)** : Add detail to the design idea specifications in order that there is sufficient information to enable estimates of the impact that each design idea will have on each performance and cost attribute. Refine the design idea specifications to the levels of detail, which reflect the 'level of uncertainty' and 'risk of deviation' from planned levels, which *you* are prepared to accept. Ensure all suspected risks, assumptions, and uncertainties are documented.

Document clearly, where any design idea has weaknesses with regard to the requirements. (For example, "Risk: Too long an implementation time." and "Risk: High risk of user dissatisfaction over usability.")

P2.6: **Consider Design Implementation**: Once you have defined the design ideas themselves, then turn *your attention to their implementation processes.* What qualifications are required for the implementers or subcontractors? What process should they follow? How will they be required to prove or measure their results? Leave nothing essential to the whims of others! Get control over your design ideas. *See 'Implementation Control' section in the design specification template in Section 7.9.*

P2.7: **Consider System Capacity and Growth**: Even when things work well in practice initially, there is no guarantee they will *continue* to do so. Success breeds volume. Volume breeds *capacity* problems. The 'good' system is no longer good enough. So we must be prepared to undertake a *continuous* responsibility for modifying the system design to meet *changed* circumstances. Sometimes 'gradual adjustment' is all that is necessary. Sometimes major new architecture is necessary. You need to be explicit about system capacity and give, if relevant, an outline of your plans of how to cater for system growth.

P3: **Evaluate Design Ideas**: We must evaluate the effects of a design on the system to which it will be added, and with regard to how it will mix with 'design changes yet to be implemented,' or even 'yet to be imagined,' by considering our long-range requirements, and architecture, for adaptability. We must ensure that we evaluate design ideas for their incremental effects on *all* of our required attributes – not just the requirements we initially designed them to primarily impact. Consider side effects – good and bad, intended and unintended.

P3.1: **Filter for Violation of any Constraint**: A design idea must not violate any applicable constraint(s). Check each design idea against each constraint. Mark the status of any design idea that violates, or potentially might violate, any constraint as 'Rejected' giving the reason in a 'Rationale' parameter. *(This is a more systematic check than might have been carried out when brainstorming during P2.3.)*

P3.2: **Estimate all Impacts**: Using an Impact Estimation (IE) table, estimate the impact of each design idea (or set of design ideas), on each performance target and each cost budget.

Cite evidence, plus/minus uncertainty and source(s) for each impact estimate. Determine the credibility of each estimate (using the credibility ratings scale from 0.0 to 1.0).

See Chapter 9, 'Impact Estimation' for further detail.

P3.3: **Consider Side Effects**: It is not just a case of checking that a design idea delivers the *required* benefits, we must also consider whether a design idea has any *unintended* negative side effects, which are *unacceptable (some negative effects may be tolerable overall).*

P3.4: **Consider Safety Margins**: You also need to assess whether the safety factors are met. Maybe a factor of two times 'over-design' is required?

If needed, return to P2 to look for further design ideas or, consider if the requirements need modifying.

P4: **Select Design Ideas and Produce Evo Plan**:

P4.1: **Initial Sequencing of Design Ideas**: We are then faced with decisions about which design ideas to select *for implementation* and which to reject. We will often be faced with several 'sufficient' alternatives; any one of which would be adequate. So how do we choose? Usually, no one dimension (for example, 'cost') is decisive.

In general, the selection decision must be made based on the *many* dimensions of measurable performance and cost. Any conditions (such as those specified in qualifiers) also impact selection.

Selection means *prioritization.* We need to determine which requirements we intend to satisfy first. *(We have an initial selection of critical requirements from P1.)* Evolutionary project management (Evo) will have the final say in determining the actual implementation sequence, but during the design engineering process we must attempt an initial sequencing of the design ideas to meet the requirements' priorities.

Establish which design ideas impact each of the constraints.

Establish which design ideas impact each target requirement (function targets, performance targets and budget targets). The impact estimation table will provide the information for the performance and budget targets. The design specifications will also hold some information depending on how much detail has been captured. *Remember the*

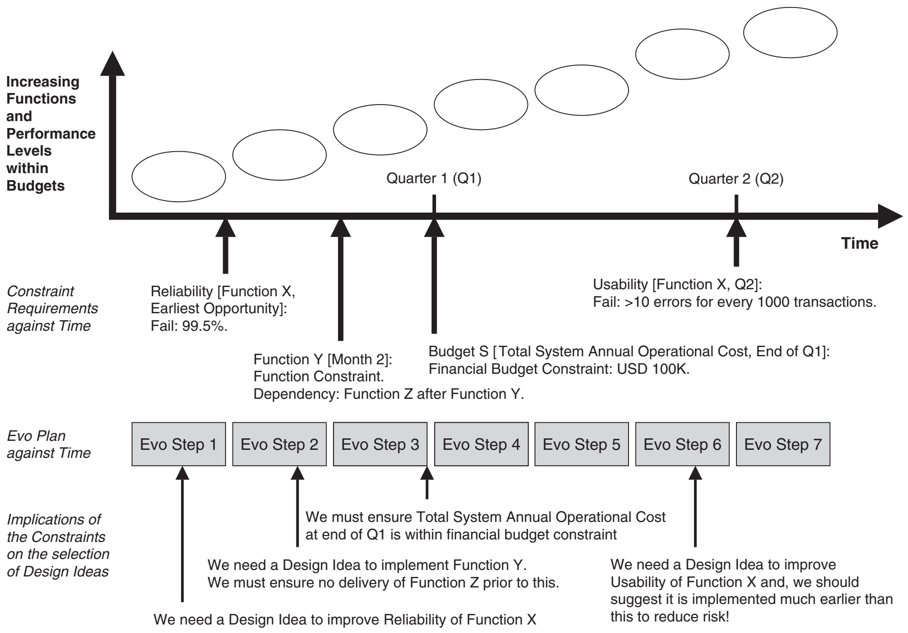

Increasing Functions and Performance Levels within Budgets
Quarter 1 (Q1)
Quarter 2 (Q2)
Time
Constraint Requirements against Time
Reliability [Function X, Earliest Opportunity]: Fail: 99.5%.
Usability [Function X, Q2]: Fail: >10 errors for every 1000 transactions.
Function Y [Month 2]: Function Constraint. Dependency: Function Z after Function Y.
Budget S [Total System Annual Operational Cost, End of Q1]: Financial Budget Constraint: USD 100K.
Evo Plan against Time
Evo Step 1
Evo Step 2
Evo Step 3
Evo Step 4
Evo Step 5
Evo Step 6
Evo Step 7
Implications of the Constraints on the selection of Design Ideas
We must ensure Total System Annual Operational Cost at end of Q1 is within financial budget constraint
We need a Design Idea to implement Function Y. We must ensure no delivery of Function Z prior to this.
We need a Design Idea to improve Usability of Function X and, we should suggest it is implemented much earlier than this to reduce risk!
We need a Design Idea to improve Reliability of Function X

Figure 7.5
Diagram shows the path for a system improving over time as Evo steps are delivered. The points marked on the time axis are the times when specified constraints have to be delivered. The sequenced Evo steps are attempting to deliver the requirements on time. The design ideas making up the content of the Evo steps have, as the first priority, to try to satisfy any constraints. Not shown in this diagram – the second priority is that they deliver the required target functions, and to the target levels for the performance and cost attributes.

performance and budget attributes do not 'hang in mid-air' – they will be attached to some functionality.

Next, identify any design idea dependencies.

At this stage, there should be a sequence of design ideas dictated by the (system scope) conditions – especially by the required timescales.

Where there are alternative design ideas, the performance to cost ratios from an IE table can be used to determine which designs contribute most efficiently towards meeting the requirements.

P4.2: **Ensure adequate Safety Margins to address Risks**: Ensure the sum of the impact estimates for each performance requirement covers the required safety margins for both performance and cost targets.

P4.3: **Consider Design Ideas with regard to the Risks**: We must also consider the uncertainties in our evaluations. The initial, purely intellectual, design process is inherently at risk of giving us false conclusions because *our* system is always somehow different from all *others*. Past design idea experience might not be valid. Our information on design effects could also be too general, or even downright wrong.

We can make allowances to cope with risk by 'over-design.' We may deliberately choose more design ideas than we strictly need in order to have safety margins. For example, we may choose two solutions that back each other up, rather than one.

We must also carefully *validate* our design choices in reality, and be prepared to *re-design* whenever practical experience shows this is necessary.

P4.4: **Consider Optimization**: Once we have an initial set of design ideas, which provide a satisfactory solution, then we can try to *optimize* using a declared optimization strategy (which might be a requirement of an engineering policy statement).

For example:

- Look for the least-cost set of design ideas, which *fully* meets the requirements.
- Select design ideas with the highest performance to cost ratios. This is *generally* good competitive practice.

But there are all kinds of variations on optimization strategies depending on your priorities regarding performance targets, resource budgets and risk (see Design Optimization within Section 7.1).

P4.5: **Re-define Design Definitions**: Re-define design ideas, if you can improve your impact estimates, and get better control over desired results by doing so. Re-define them so that they have substantially different performance and cost impacts, in the direction you need them to be.

P4.6: **Consider Pilot and/or Trial**: Plan to try out design ideas, which *seem* high risk, in pilots and/or trials, or specify them for implementation in early evolutionary result cycles. Feedback any results into *this* process (at P3.2: Estimate all Impacts.). Refine your estimates.

Design knowledge, from past uses of a design, gives us some idea of how we can expect things to work. But, new systems contain many new elements of technology, inputs and people. Thus, only practical use of a new design idea, in the real environment, will assure us that we were correct in our estimates of design effect or will convince us that we were to some degree wrong. So, the design engineering process must somehow be linked with a practical process of trying things out, well before large-scale irreversible commitment is made to design ideas.

Exit Conditions

X1: The Generic Exit Conditions apply. The set of design specifications should have exited SQC with no more than one estimated remaining major defect/page. *(Expectation: If you don't demand such* *a low exit level, the specification will have 20 or more major defects/page.)*

X2: A safety factor of <*four*> is required for all performance and cost attributes.

Note: (4 times over-design, is NOT 4 times more cost!)

Note: Process 'Exit' means we can *then* use the design specification for planning *trials to get feedback* (that is, using the design in Evo steps, see Chapter 10). It does not mean the design specification is a 'final' specification.

Gap Analysis by Igor H. Ansoff

The procedure within each step of the cascade is similar.

(1) A set of objectives is established.
(2) The difference (the 'gap') between the current position of the firm and the objectives is estimated.
(3) One or more courses of action (strategy) is proposed.
(4) These are tested for their 'gap-reducing properties.'

A course is accepted if it substantially closes the gaps; if it does not, new alternatives are tried.

Igor H. Ansoff, Corporate Strategy *(Ansoff 1965 Pages 25–26). Also quoted in Mintzberg (1994).*

7.6 Principles: The Design Engineering Process

1. **The Principle of 'Design Ideas are only as Good as the Requirements Satisfied'**
   Design ideas cannot be correctly judged or validated *except* with respect to *all* the performance and cost requirements they must satisfy.

2. **The Principle of 'The Best Chess Move'**
   You should try with each increment of design specification or design implementation, to get the best possible satisfaction of your unsatisfied performance requirements, from your unused cost budgets.

3. **The Principle of 'Results Beat Theory'**
   Design ideas are only as good as their *real results*, not their *intent.*

4. **The Principle of 'Early Surprises'**
   You never *know* how it works, until you have actually tried out a design idea in practice. Get surprised as early as possible!

5. **The Principle of 'It's Not Just What You Do, It's How You Do It'**
   Design ideas must try to exercise control over both design *content* and design *implementation*. The devil is in the details!
6. **The Principle of 'Good is Not Always Good Enough'**
   A 'good' design idea might not be good enough to meet all *your* targets on *time*.
7. **The Principle of 'Designs should have Good Return on their Investment'**
   'Good' design ideas might cost *too much*, sooner or later.
8. **The Principle of 'Sneaky Gremlins'**
   Apparently 'good' design ideas might have subtly-hidden nasty side effects. Estimate them, know when you don't know them, measure them, and don't assume they won't hurt you! They will show *you* no sympathy!
9. **The Principle of 'Design Beats Test'**
   Design performance 'in', and design 'to control' costs:
   You cannot *test* quality *into* a badly designed system.
10. **The Principle of 'Eternal Vigilance for the Butterfly Effect'**
    You never *finally* know about a design idea's effects;
    Tomorrow's slightest change might ruin your whole project.
    Even initially successful designs might have to be adjusted for growth and change.

7.7 Additional Ideas

Priority Determination

Systems engineering can be viewed as a constant stream of priority evaluations. So priority determination is a key concern. However, the conventional means of deciding priority are frequently inadequate: a subjective weighting approach is, unfortunately, often adopted. Ideally priority determination for implementing requirements should be:

- a 'performance to costs impact' and 'resource-focused' process. It should consider value to cost ratios, return on investment (ROI) and take into account resource availability.
- an information-based process, which makes full use of the available factual information, and is able to reuse this information. Not a weight-based process.
- a dynamic process, which uses feedback from the ongoing implementation; and is open to instigating, and catering for, change in requirements and in design ideas.

Ideally the ultimate values to the stakeholders, which are the results of the system performance characteristics, would be evaluated and used to determine priority. In practice it might be difficult for a systems engineer to access the stakeholder domain data needed to calculate the value that the stakeholder would expect to experience. Even the stakeholder might have difficulty estimating the 'value delivered' accurately. So, we might choose to fall back on a more immediate notion of stakeholder value – meeting the required targets.

What is wrong with the Subjective Weighting Approach for Determining Priority?

In the priority weighting (or priority ranking) process, each element of a set of elements in a decision-making model, is subjectively assigned a numeric value indicating its priority (For example, a value on a scale of 1 to 10 or, a percentage weight).

The degree of subjectivity[4] is determined by such factors as the actual people asked (the number of people, their roles and their expertise) and how they arrive at their decisions (their decision *processes*; including such things as their influences). In many cases, people are asked on a one-off basis during a group meeting to assign numerous comparative weightings 'off-the-top-of-their-heads.' Inadequate documentation of who, when and why (experience and/or fact) is widespread. The reasons why such a process is weak, when determining the priority for implementing requirements, include:

- Information overload: too many things have to be taken into account at once for subjective assessment to work well.
- Lack of specific information: often there are gaps in the information available: evidence and source data are usually missing.
- 'One-off' weightings: weightings tend to be 'frozen', they are not reassessed frequently.
- Lack of consideration of resources: resources are simply not taken into account.
- An individual stakeholder's viewpoint is limited (a person's subjective judgment depends on many things. For example, experience and access to information).
- Typically people can only participate in supplying their requirements and committing their resources. They are unlikely to be able to make a globally optimal priority decision, on behalf of the entire stakeholder community.
- In a group meeting, factors such as authority, office politics and personality interfere with the outcome.

[4] Note, I am objecting to subjective weightings, not to stakeholders proposing their own subjective requirements.

The Role of Resource in Determining Priority

Planguage defines priority as follows:

> A 'priority' is the determination of a relative claim on limited resources. Priority is the relative right of a competing requirement to the budgeted resources.[5]

If resources were unlimited, there would be no need to prioritize things. You could have it all.

Many approaches to priority oversimplify or even eliminate consideration of resources (for example, see Saaty 1988; Akao 1990). Yet return on investment (ROI) is ultimately the key driver when deciding priority. What value shall be obtained in relation to the 'resources needed' (the costs)?

Resource availability can also be a factor in determining implementation priority. Selection of a priority solution might be:

- influenced by a lack of some resources
- affected by the ability to substitute one resource with another.

Planguage Information Supports Priority Determination

Planguage captures a wide range of reusable information that supports priority determination. It quantifies all scalar requirements and caters for individual deadlines at a detailed level, and as a result gives you a greater level of priority control. Some key Planguage specification parameters assisting priority determination are as follows:

- Value
- Stakeholder
- Constraint
- Target
- Dependencies
- Qualifiers [Time, Place, Event]
- Authority
- Source.

The source, authority, and stakeholder information establishes the stakeholders affected by a requirement, and their level of responsibility. When determining priority, meeting any constraints is the first priority. The next priority is to meet the targets. The qualifiers narrow the requirements down to the specific conditions: time qualifiers specify

[5] I mean all types of resource including time to deadline, human effort, money and space.

the timescales, place qualifiers limit the system space and event qualifiers state any specific circumstances, which apply.

Planguage might capture this information, but it requires evaluation to establish the priorities. There may be priority conflicts needing negotiation.

Priority Strategy

One piece of information vital for priority determination is the strategy for priority. There are several different strategies that could be chosen. See discussion on Design Optimization within Section 7.1.

Dynamic Priority Evaluation

Planguage adopts a dynamic, numeric idea of priority. Priority is defined as the claim on resources to develop or operate a system. It is the currently unfulfilled requirements, (the gaps) which have priority. Our highest priorities, at any moment in time, are the unfulfilled requirements that are due next, date-wise.

There are no artificial weighting factors needed in Planguage. We use only direct natural statement of the qualities and costs we want, together with when we want them. We compare 'what we want' with 'what we have' at the moment. The larger the gap between 'wants' and accomplishments, the higher the current priority in that area to do more design work or to do more implementation work.

Using Evo, priority control becomes early, frequent and continuous, throughout the project design and implementation phases. Priorities change as they are satisfied (just as appetite changes as food satisfies it).

Also, the basic requirements can change at any moment of a project. It would be convenient if they didn't, but the real world is not that co-operative! Continuous re-assessment of priority, allows any changes in the requirements to be incorporated into the system design process.

See also Section 9.7 on priority.

7.8 Further Example/Case Study: Design Specifications Masquerading as Requirements

This is a sample of some real design ideas, which were *found in a requirement specification.* (Certain details are changed for confidentiality.)

They are extremely early outline drafts and still need a lot of work! We certainly had not yet enhanced the specifications to the level required by the Planguage template in this chapter. However, the drafts do give some practical insights into simple Planguage formatting. The most important steps we took were as follows:

- to refuse to treat them as requirements
- to identify the performance attributes they were *intended* to impact (*see 'Impacts' parameters below*) and
- to define the impacted performance attributes properly.

EXAMPLE

Adaptive Channel Allocation: ACA: Design:
Assumption [ACA]: New Product must automatically yield to Macro cellular system, and to re-tuning of the Macro radio network.
Impacts [Co-existence]: Slow or Fast? <- Marketing Specification 3.11.
Note: This is one design idea, not a constraint.
Automatic Roaming Designs: "A rough collection of design ideas."
Impacts: Automatic Roaming.
Note: these may be design constraints! <- New Product Team 4 March.
IS-41 signaling link to the public network <- Marketing Specification 5.2.1.
Signaling, data and messaging interfaces <- Marketing Specification 5.2.
The New Product must support the protocol of Cellular Messaging Teleservice (CMT) over its signaling link over both public and cellular network <- Marketing Specification 5.2.2.
The New Product must support the receipt and acknowledgment protocol for voice-message-waiting indication <- Marketing Specification 5.2.3.
Cell Plan Minimization:
Impacts: {Installability, Maintainability}.
Cooling Fans [**Radio Heads**]: to be avoided to avoid noise, but quiet ones, as defined by Quietness quality requirement, acceptable.
Impacts: Quietness <- Marketing Specification 4.1.5.
Product Evolution: Design Idea. "These design ideas are a rough collection from the Marketing Specification".
Impacts: Evolution.
New Product shall have a modular structure <- Marketing Specification 3.2.
Modular, future proof <- Marketing Specification 4.3.3.
New Product shall be easy to upgrade <- Marketing Specification 3.3.
The switch must support remote SW loading <- Marketing Specification 4.3.4.
'Plug & Play' <- Marketing Specification 4.5.2.
Remote Software Upgrade: for both correction and upgrading proposes <- Marketing Specification 4.5.14.
Software changes shall not require manual physical access to Radio Heads <- Marketing Specification 4.5.15.
New software – upgrades, patches, new releases, etc. should require a minimum of scheduled downtime for New Product <- Marketing Specification 4.5.16.
Home Location Register: HLR:
HLR is part of the Macro cellular system?? <- Marketing Specification 4.3.7.
Impacts: <unspecified>.
Low Power Consumption [**Radio Heads**]:
Low Power consumption will be designed.
Impacts: Quietness "in order to avoid fans and consequent noise" <- Marketing Specification 4.1.5.

EXAMPLE CONTINUED

Low RF Power Output [**Radio Heads**]:
Impacts: {<avoiding interference>, Availability, Co-existing, Per User Cost, Robustness, others} <- Marketing Specification 4.2.1.8.
Remote SW Loading:
The switch must support remote SW loading <- Marketing Specification 4.3.4.
Impacts: Maintenance.
Single Cabinet [**Central Equipment**]:
The Central Equipment must fit into a single cabinet including power, but not batteries <- Marketing Specification 4.5.4.
Comment: This is really a way to achieve Volume of 36 liters as estimated by TW.
RH1: Assumption: RH assumed to be single cabinet.
Impacts: <unspecified>.
Basically, what we did was to identify these design specifications as design ideas, not requirements (design constraints), and to structure them so we could see their Source and their Impact intents.

7.9 Diagrams/Icons: The Design Engineering Process

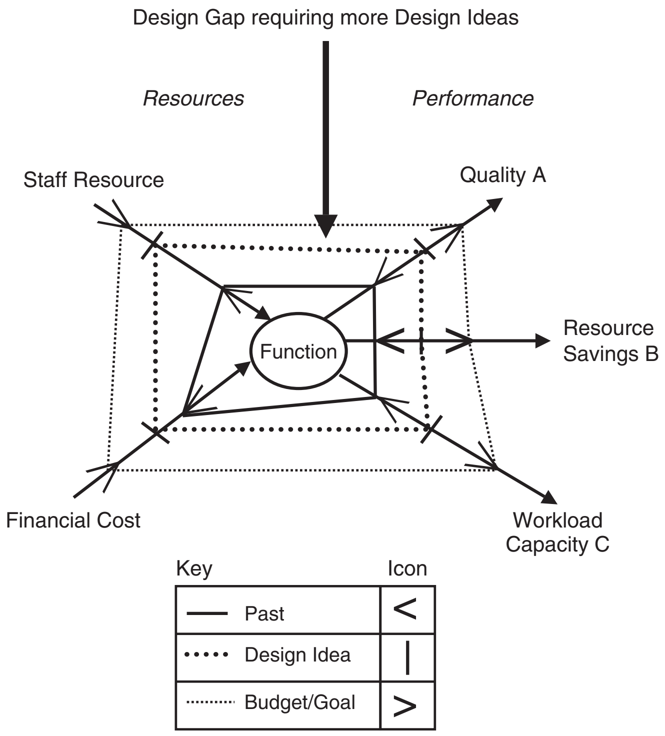


Figure 7.6
Diagram showing the gap between the Past and the Goal/Budget levels and the contribution that a Design Idea makes towards filling the gap.

Design Specification Template <with Hints>

Tag: <Tag name for the design idea>.
Type: {Design Idea, Design Constraint}.

=========================== Basic Information ===========================
Version: <Date or version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: < Role/e-mail/name of person responsible for changes and updates>.
Expert: < Name and contact information for a technical expert, in our organization or otherwise available to us, on this design idea>.
Authority: <Name and contact information for the leading authorities, in our organization or elsewhere, on this technology or strategy. This can include references to papers, books and websites>.
Source: <Source references for the information in this specification. Could include people>.
Gist: <Brief description>.
Description: <Describe the design idea in sufficient detail to support the estimated impacts and costs given below>.
<Term Tag here>: Definition: <Use this to define specific terms used anywhere in the specification>. "Repeat this for as many definitions as you need"
Stakeholders: <Prime stakeholders concerned with this design>.

========================== Design Relationships ==========================
Reuse of Other Design: <If a currently available component or design is specified, then give its tag or reference code here to indicate that a known component is being reused>.
Reuse of This Design: <If this design is used elsewhere in another system or used several times in this system, then capture the information here>.
Design Constraints: <If this design is a reflection of attempting to adhere to any known design constraints, then that should be noted here with reference one or more of the constraint tags or identities>.
Sub-Designs: <Name tags of any designs, which are subsets of this one, if any>.

========================= Impacts Relationships =========================
Impacts [Functions]: <List of functions and subsystems which this design impacts attributes of>.
Impacts [Intended]: <Give a list of the performance requirements that this design idea will positively impact in a major way. The positive impacts are the main justification for the existence of the design idea!>.
Impacts [Side Effects]: <Give a list of the performance requirements that this design idea will impact in a more minor way, good or bad>.
Impacts [Costs]: <Give a list of the budgets that this design idea will impact in a major way>.
Impacts [Other Designs]: <Does this design have any consequences with respect to other designs? Name them at least>.

======================= Impact Estimation/Feedback =======================
For each Scalar Requirement in Impacts [Intended] (see above):
Tag: <Tag name of a scalar requirement listed in Impacts [Intended]>.
Scale: <Scale of measure for the scalar requirement>.
Scale Impact: <Give estimated or real impact, when implemented, using the defined Scale. That is, given current baseline numeric value, what numeric value will implementing this design idea achieve or what numeric value has been achieved?>.
Scale Uncertainty: <Give estimated optimistic/pessimistic or real ± error margins>.
Percentage Impact: <Convert Scale Impact to Percentage Impact. That is, what percentage of the way to the planned target, relative to the baseline and the planned target will implementing this design idea achieve or, has been achieved? 100% means meeting the defined Goal/ Budget level on time>.

Figure 7.7
Continued next page

Percentage Uncertainty: <Convert Scale Uncertainty to Percentage Uncertainty ± deviations>.
Evidence: <Give the observed numeric values, dates, places and other relevant information where you have data about previous experience of using this design idea>.
Source: <Give the person or written source of your evidence>.
Credibility: <Credibility 0.0 low to 1.0 high. Rate the credibility of your estimates, based on the evidence and its source>.
===================== Priority and Risk Management =====================
Rationale: <Justify why this design idea exists>.
Value: <Name [stakeholder, scalar impacts and other related conditions]: Describe or quantify the knock-on value for stakeholders of the design impacts>.
Assumptions: <Any assumptions that have been made>.
Dependencies: <State any dependencies for this design idea>.
Risks: <Name or refer to tags of any factors, which could threaten your estimated impacts>.
Priority: <List the tag names of any design ideas that must be implemented before or after this design idea>.
Issues: <Unresolved concerns or problems in the specification or the system>.
======================== Implementation Control ========================
Supplier: < Name actual supplier or list supplier requirements>
Responsible: <Who in your organization is responsible for managing the supplier relation?>
Contract: <Refer to the contract if any, or the contract template>
Test Plan: <Refer to specific test plan for this design>
Implementation Process: <Name any special needs during implementation>
======================= Location of Specification =======================
Location of Master Specification: <Give the intranet web location of this master specification>.

Figure 7.7
Design Specification Template. This is a form to fill out, with <hints in fuzzy brackets>.

7.10 Summary: The Design Engineering Process

The 'design engineering process' is a systematic, rational process of finding design specifications, which when implemented will satisfy a balanced set of requirements on time.

The term 'design engineering' means a design process based on multi-dimensional quantified requirements and multiple quantified design attributes. It requires concurrent use of an implementation process, like Evo, based on quantified measurement of performance and costs at frequent evolutionary cycles, and of necessary analysis and correction to maintain progress towards (potentially adjusted or traded off) formal and quantified targets.

The selection of design ideas is determined by the need to deliver a set of specified stakeholder target levels within a set of specified constraint levels.

The design engineering process is really concerned with identifying optional design ideas and evaluating the alternative possibilities to find a satisfactory architecture (that is, the sum of all design ideas), which provides:

- the best fit with the requirements
- early delivery of key results (with high stakeholder values)
- best value to cost ratios and performance to cost ratio, and
- acceptable risks.

The design engineering process may also involve identifying the best reaction (redesign) to any feedback (good or bad feedback from actually implementing design ideas in the real system).

The design engineering process cannot usually be done, competitively, in a single pass. The effects of even a single design idea are too complex to understand without 'experience analysis' from past use of the idea (see 'Impact Estimation', Chapter 9), and especially without actual use on our new system (see Evo, Chapter 10). So it must normally be expected that the 'final' and 'correct' design specification can only be evolved towards (never perfectly or ideally reached) as a result of multiple feedback-and-change cycles.

Refinement of design can be done in parallel with actual use (by at least some early stakeholders) of a version of the product. The practical feedback from this early delivery can be used to improve the design; probably faster and more correctly than by staying in the 'design phase' longer.

A further complication is that as time goes on, both the 'design requirements' and 'potential and selected design technology' will 'expectedly' change, thus requiring yet another set of cycles of learning how to satisfy these new, changed requirements. Never perfect, continuously better, is the watchword.

In terms of 'Competitive Engineering' you can *always* refine the design to be more competitive. However, there is a point where the cost and time of refining the design exceeds any competitive benefit, and it is time to stop designing and to get the product out of the door, this time around.

Design Policy

Design ideas are only really finally validated when they display satisfactory attributes in a real system (that is, after successful delivery in an evolutionary step). Don't kid yourself that they are 'final' before that.

A suggested mental attitude towards design specifications. Don't believe any estimates of performance and cost, only reality as measured!

Chapter
8

Specification Quality Control

How to know how well you specified

Glossary Concepts

Specification Quality Control (SQC)
Defect Detection Process (DDP)
Defect Prevention Process (DPP)
Specification
Source
Kin
Checklist
Issue
Major Defect
Minor Defect
Checking Rate

8.1 Introduction: Specification Quality Control

Specification Quality Control (SQC) is the name I shall use to refer to this method in this text. Within the software community, the term 'Inspection' is used. However, it is a poor choice for engineering communities, which already use 'inspection' in another sense during final production line quality control. SQC is remote from such assembly-line inspections, as it takes place from the earliest stages of idea specification and has different organizational impacts (for example, team building and assisting in 'on the job' training).

The primary purpose of SQC is systems engineering process control through sampling measurement of specification quality. Through SQC, we can improve systems engineering processes, save project time and increase systems engineering productivity.

Improving Process

Control of projects, designs, strategies, marketing, selling and buying, management planning, and programming, all have one thing in common at least – they rely on ideas *specified* by people, and *read* by people. If those ideas are misunderstood by the reader, incomplete, wrongly written or out of date, then we are doomed to lose control and be less competitive, no matter how well we design, plan and implement!

For *software*, studies have long since shown that a considerable percentage (44% at Bellcore (Pence and Hon 1993) and 62% (Thayer, Lipow and Nelson 1978)) of all bugs in computer programs were not due to faulty programming. They were due to faulty requirements and design being handed to the programmers and the testers. In many cases, the testers, unwittingly, checked that an erroneous specification was 'correctly' programmed! Testing, in this situation, does not solve the problem: it confirms it. However, SQC can address such problems.

In *aircraft design* at Douglas Aircraft (now Boeing), 'engineering order' faults cost $2,965 each to correct and 30% of engineering orders needed correction. After SQC was applied in 1987–88, the percentage of faulty engineering orders fell to 0.5% (Personal Experience). We achieved similar results in 1989 at Boeing, Renton on all aspects of aircraft design.

The tendency to commit some kind of error, when communicating complex ideas in writing to other people, is much worse than most people realize. My own experience in industrial measurement of defects suggests that technical documents, initially and routinely, contain at least 20–60, and often far more, 'major' engineering specification defects in each 'logical' page (300 non-commentary words). Through systematic use of feedback from SQC to specification writers, this level can be brought down to well under one remaining major defect/page (British Aerospace, Eurofighter Project, Wharton, achieved this in 18 months (Personal Communication)).

Saving Time

Without SQC, a major defect left in a technical specification can cost an average of 9.3 work-hours to deal with.[1] Use of SQC at an early stage (during writing the specification) would cost only *one* work hour to remove it. "A stitch in time saves nine" (or an SQC hour saves nine-point-three to be exact! (Gilb and Graham 1993)).

Increasing Productivity

The reduction of defects (as a result of using SQC) saves 'rework', which is otherwise about half of all effort in software projects. Raytheon (Haley et al. 1995) found that software engineering productivity for about a thousand programmers increased by a factor of 2.7 over a few years of using SQC (Inspection and Defect Prevention Process).

One major reason for defect reduction is the 'training effect' of SQC on individuals. The number of defects injected by a systems engineer reduces by about 50% each time they go through an SQC process (Personal Experience since 1988). Systems engineers rapidly learn to take the rules seriously. They see that their peers expect them to comply with the rules and that their work cannot exit, and be 'finished', until they reach at least the exit level for the estimated number of remaining major defects. I have found that this is as true in software as it is in hardware engineering.

Industrial Usage

The methods needed for quality control (QC) of specifications originated in the early 1970s within IBM, when they were used under the name of 'Design and Code Inspections'.[2] Since then significant changes have occurred, resulting in the SQC method described in this book. The most notable change was the introduction, again within IBM, of the Defect Prevention Process (DPP) (Mays 1995). The other major change is the shift to 'sampling', rather than 100% checking and trying to clean up defects.

[1] As measured on a 1,000 defect sample by (then) Thorn EMI (electronics industry) in 1990. See Section 8.8 and (Gilb and Graham 1993 Page 315: Reeve).

[2] Fagan, M. E. 1976. Design and code inspections. *IBM Systems Journal*. Volume 15. Number 3. Pages 182–211. Reprinted 1999. *IBM Systems Journal*. Volume 38. Numbers 2 and 3. Pages 259–287. See http://www.research.ibm.com/journals/sj/

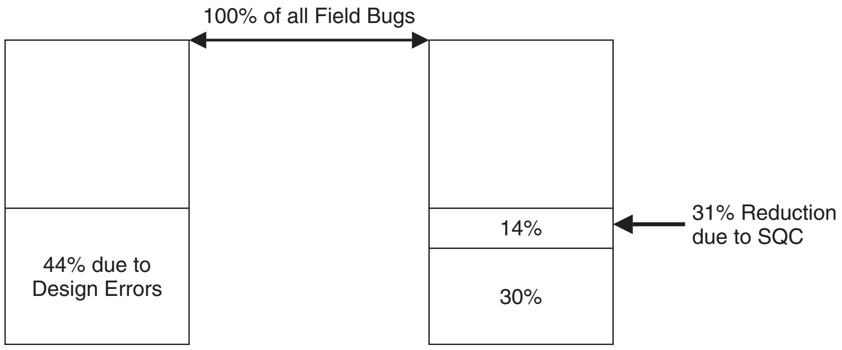


Figure 8.1
Due to use of SQC during development of telecommunications software, a 31% reduction in design errors that caused bugs in the field was measured after 2 releases (Pence and Hon 1993).

The first large-scale *hardware* engineering uses of SQC took place at Douglas Aircraft (1988) and Boeing (1989) under this author's guidance. In recent years, Siemens, Alcatel and Ericsson have also successfully used the method on a large scale (hundreds trained) for total product development purposes. Hewlett Packard has reported estimated savings due to SQC (some use within hardware product planning) of $21.5 million and $34 million in 1993 and 1994 respectively (Grady and Van Slack 1994).

The use of SQC *outside* of the software area is, as yet, little understood or appreciated, except by the few corporations who have tried it out such as Ericsson, Douglas and Boeing. It is time that this industrial experience was more widespread knowledge. There is little difference in the specification of software engineering, management planning or hardware engineering with regard to human specification errors, their causes and their consequences.

8.2 Practical Example: Specification Quality Control

Take the simple performance requirement statement:

'The objective is to get higher adaptability using modular structure.'

Do you see any problems with it? Is it similar to statements you see every day? Well, if you have read this book this far, you would notice that it violates some rules we have suggested. Of course, there is nothing wrong with it, unless we agree that these rules are in force. For some purposes they should be in force, for others not.

SQC works by using the formal rules that are in force: a 'defect' is a rule violation. SQC discovers whether people have applied the agreed rules or not. A specification writer must always know the rules that apply (and have agreed to them in advance). The specification writer should welcome any help to follow them. Rules, after all, should be 'best practice' rules.

Let us now (for the sake of this example) introduce a few short rules, which apply to the quality requirement statement above.

Rules For Performance Requirements

Tag: Rules.OBJ.

Clear: They must be *unambiguously* clear to the intended readers (not to 'anyone,' just the relevant people).

Detail: They must detail complex concepts as a set of elementary measurable concepts.

Scale: They must specify a scale of measure to define the concept (all performance attributes are quantifiable).

Quantify: They must specify at least two points of reference on the defined Scale to define 'relative' terms, such as ''higher.'' These are called the benchmark and target specifications.

Qualify: Targets must specify exactly 'when' a performance level is to be available. Other qualifier notions, such as 'where' and 'if' should also be made explicit, if the target is not elsewhere specified.

Ends: They must not put 'designs' in the specification of 'performance requirements.' Specify the *Ends*, not the *Means*.

Source: The source statements for each requirement must be precisely referenced (for example, <- the contract and marketing documentation).

Fuzzy: Fuzzy unclear concepts shall be marked with <fuzzy/angle brackets> to indicate there is room for improvement.

A checker (a person assigned to check a specification and its selected source documents against these rules) would be obliged to report, for the performance requirement statement about 'higher adaptability', that all the above rules were violated.

There are, therefore, at least eight defects in the requirement statement. If these defects *might* have much higher costs later in a project (if not fixed at specification time), they should be classed as 'major' defects. Majors are the defects it pays to fix now, at a tenth of the cost we would otherwise suffer later. (Fixing majors early is useful, but preventing their injection is even more profitable.)

Checkers are friendly, confidential personal advisors to the specification writer. The checker's first job is to point out potential problems for correction before a specification is released to other engineers, or to customers. Checking is a service the writer will likely perform, in return, when their former peer checkers specify something themselves, and want SQC help. The responsible engineer will take a list of the checker's suggested advice regarding 'potential defects' (issues) and consider correcting them. They should address similar defects, outside the sample checked, as far as necessary, according to the applicable rules, procedures and source documents. However, it may pay off to totally rewrite the specification. The specification document 'Exit Level' is based on a general calculation of what is the best project time-saver. We don't exit, if cleaning up the specification now saves the most time, in the long run. **The following are the expected results of a *single pass* of SQC:**

(Note: Multiple passes should be rare.)

1. Based on defects found and corrected and, on an assumed SQC effectiveness at spotting defects of 50%, a calculation will be made about the (probable) remaining major defects in the specification (which is about as many defects as we found – since we cannot expect to be much better than 50% effective in finding defects). If these are more than permitted by the exit conditions, the specification will not be released. This is because the estimated unfound remaining majors would cause more loss of time than savings to be gained, if we let them exit downstream; that is, if we released the specification immediately.
2. The specification writer will learn about current agreed rules and their peers' interpretations of these rules. As a result they are likely, by my industrial experience, to learn to produce a specification with half the number of defects next time. (Ultimately, after several SQC experiences for the writer, about 100 times cleaner – using major defect reduction as the measure – specification is usually achieved!)
3. The checkers themselves will learn best practice rules and their peers' attitudes towards those practices. This will influence the checkers' specification work quality.
4. The 'users' of the specification will learn to expect (in terms of their entry condition) a minimum specification quality level (such as no more than one remaining major defect/page).
5. The SQC team will continuously suggest process improvements to reduce future major defects. (Poor working processes, training, tools and the working environment 'force' defects on the workforce according to Deming (Deming 1986)).
6. Project productivity will at least *double*, mainly due to fighting fewer defects later (Dion has reported productivity increasing by a factor of 2.7 (Dion 1993; Haley et al. 1995)).

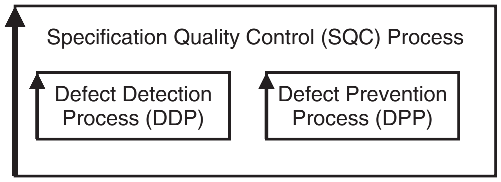
As a result of SQC we will have data to decide if it pays off to release the specification to another engineering process, or fight the defects now.

8.3 Language Core: Specification Quality Control

Basic Definitions (see also Glossary terms)

Specification Quality Control (SQC)

Specification Quality Control (SQC) consists of two main processes: the Defect Detection Process (DDP) and the Defect Prevention Process (DPP).


Specification Quality Control (SQC) Process
Defect Detection Process (DDP)
Defect Prevention Process (DPP)

Defect Detection Process

The Defect Detection Process is concerned with document quality, mainly with identifying defects in the documentation and using this information to make decisions about how best to proceed with the main document under SQC – the main specification.

Ideally, though sometimes not done due to the economics of the situation, a known defect must be removed as soon as possible after the error has been committed. This is to avoid the high cost of late removal (at test or in field) of the defect, or to avoid the high cost of its consequences. "*A stitch in time saves nine.*"

Defect Prevention Process

The Defect Prevention Process is concerned with learning from the defects found and suggesting ways of improving processes to prevent them reoccurring in future. The process improvement suggestions are routed on to the relevant process owner for further consideration. "*An ounce of prevention is worth a pound of cure.*"

Here are some other basic SQC concepts.

Issue

An issue is a *perceived* defect in a document. It is a non-confrontational way for a checker to say, "I think I may have identified a defect."

Defect

A failure to observe a formal, written, required rule. It is not a personal opinion or personal taste. It is failure to observe a group norm, or required best practice.

Major defect

A major defect is a defect (rule violation) which, if not fixed at the requirements or design stage of specification, will possibly grow approximately an order of magnitude or larger in 'cost-to-find-and-fix' and/or damage potential. It is often intentionally written with a capital 'M'. Minor defects tend not to be economic to identify or fix (*but you sometimes have to identify them to determine that they are indeed minor and not, major*).

Page

A logical page, as opposed to a physical page, is defined as a specific number of non-commentary words. If no other definition is given then use '300 non-commentary words' for each logical page (default 'volume' definition). This ensures measurements of checking rates and defect densities are consistent.

Checking Rate

The checking rate is the average speed with which an individual checker searches a specification for defects, allowing time for checking it against rules, sources, kin documents and checklists. This is a critical factor to control for effective checking. You have to go surprisingly slowly to raise your checking effectiveness from 5% to 50%. (For example: one page an hour!)

Optimum Checking Rate

The optimum checking rate is the rate, which gives the highest checking productivity (effectiveness in finding majors). It is the checking speed that in fact works best on a given document type for an individual checker to do their assigned tasks. It is found by establishing the most effective average historical checking rate in terms of finding major defects. The optimum checking rate is usually in the range of 300 non-commentary words/hour (plus 300/minus 270). This is used as a guide for team planning. Individuals need to tune in to their personal optimum rate, which varies from this average.

The major trick to going at this 'slow' rate is to *sample*, not to attempt 100% checking of all pages and consequent 'defect clean up'.

Remaining Major Defects

The remaining major defects are the *estimated* remaining major defects/'volume' measure (which could be a page, a technical drawing or an entire specification) given for a sample or an entire specification. It is estimated based on 'total found' and 'known % effectiveness.'

Checklist

A 'checklist' is a list of questions, which can be asked about a document's contents by a checker, with a view to improving the effectiveness of that checker in finding major defects. Checklist questions are always directly derived from individual official rules. They are not allowed to be the rules, or to change the rules, just to *interpret* them.

Rule

A rule is a standard for the production of a *written* process output. A rule can be used to judge the objective quality ('defect-freeness' according to current rules) of a written process output. Violations of rules are defined as 'defects.'

Rules are often grouped into sets according to the type of standard, which they are setting (for example, 'specification clarity' or 'specification consistency').

Main Specification

The main specification is one of potentially *many* documents involved in a single SQC. However, it distinguishes itself as the one we are trying to get formal 'exit' for. Exit (acceptable exit level) is based primarily on the specification's quality with respect to the official systems engineering standards (rules) for writing it.

Source Documents

The source documents are the 'parents' used to produce a specific main specification. For example, contracts are typical sources for requirements. Requirements are a source for design. Requirements and design are sources for Impact Estimation. Design is source for planning, estimating and construction or programming. Change requests are sources for an updated specification.

Kin Documents

'Offspring of the same 'parent' (source) documents are 'kin.' *For example, test plans, source code and user handbooks could all be derived from the same requirements or the same design.* The use of kin documents is that they can serve as information to perform defect checking in SQC.

8.4 Standards: Specification Quality Control

Rules are standards, and are central to the SQC process; specifications must be checked against their agreed specification rules. However, the rules to be used depend on the specification type, so we won't attempt to list them here. The rules given in other chapters of this book are suitable examples of such rules (*but they are by no means a complete list*).

Here is a list of guidelines for assessing whether your overall SQC process is functioning correctly.

Guidelines for assessing functioning of overall SQC

Economic: SQC must always make economic sense. If SQC is not saving in the order of 10 hours for every hour spent on SQC, then your SQC process should probably be modified or abandoned.

SQC Champion: There must be an SQC champion within the organization. (At the very least, a nominated person responsible for SQC; an SQC process owner.)

Team Leaders: There must be a list of current SQC team leaders. It should show that there is a sufficient number of team leaders within the organization and also that the team leaders are trained, tested and 'certified' to ensure they know what they are doing.

Statistics: The SQC statistics must be up-to-date on the SQC database.

Meetings: All meetings must be of maximum length of two hours (tiredness reasons). If more time is needed, schedule a set of such meetings (*but do consider the possibility of using sampling*).

Checkers: Unless you are training novices, the number of checkers at a meeting should be five or less. Two or three people is typically most cost-effective, Four to five is generally more 'effective.'[3]

Checking Rate: All checking must be carried out near the relevant optimum checking rate. This rate will vary by document type and organization. It is about 1 page/hour.

Condition: Entry and exit conditions must be taken seriously. They are there to save you wasting time. The number of remaining major defects/page for successful exit must be ultimately less than one (major defect/page).

Standards: There must be an up-to-date (intranet) 'library' of official rules, checklists and forms.

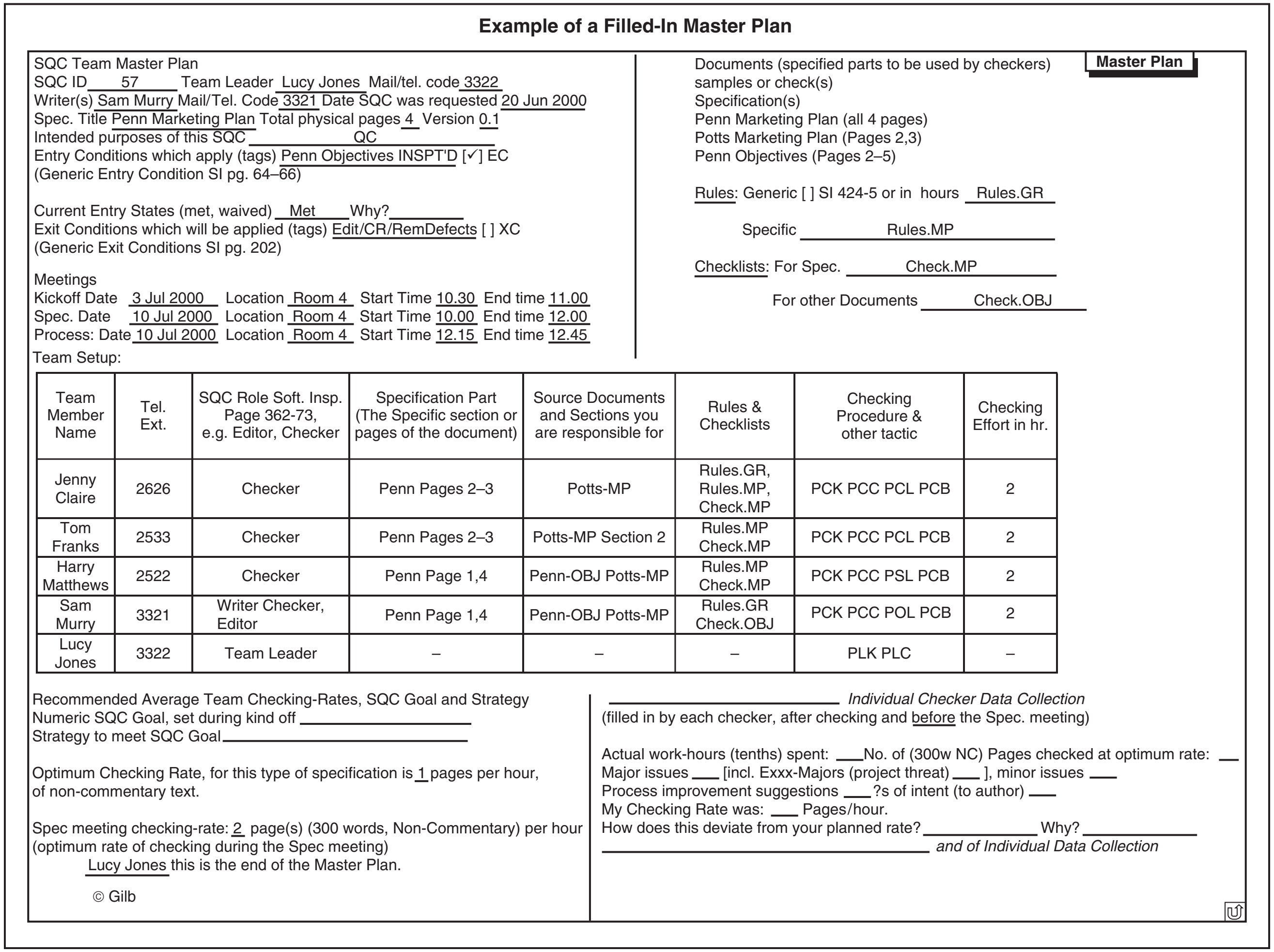
Upstream Pollution: The number of major issues identified by your team in source specifications, which have just previously-exited SQC, should be approximately 15% of the total number of logged issues. Otherwise, this is a sign that your team is not taking the 'second-round' opportunity to find source defects, seriously.

Forms

SQC uses four main forms: the Master Plan, the Editor Advice Log, the Data Summary and the Process Meeting Log. There are examples of these forms filled in, in Figures 8.2, 8.3, 8.4 and 8.5. Blank forms are given in Section 8.9.

Note forms are a 'procedure' (in the format of the form) for gathering data. Most of our clients have their own local variation of the forms and automate them (usually on an intranet web site).

[3] The original evidence for this came from research performed by Søren Nielsen in the Danish electricity industry (Danish Technical Institute, Lyngby, 1987; cited in Gilb and Graham 1993), and was confirmed by further research at Jet Propulsion Labs by John Kelly (Kelly 1990a; 1990b). Optimum effectiveness (number of unique issues per checker) was achieved with teams of 4–6 people, optimum efficiency (cost per unique issue found) with teams of 2–4 people. The recommended team size of 4–5 people achieves the best compromise between these factors. It was Edward Weller, analysing data from more than 6,000 inspection meetings conducted at Bull HN (Weller 1993), who reported that "four-person teams were twice as effective . . . as three person teams." Also included in Wheeler, Brykczynski and Meeson (1996).



Example of a Filled-In Master Plan

Master Plan

SQC Team Master Plan
SQC ID 57 Team Leader Lucy Jones Mail/tel. code 3322
Writer(s) Sam Murry Mail/Tel. Code 3321 Date SQC was requested 20 Jun 2000
Spec. Title Penn Marketing Plan Total physical pages 4 Version 0.1
Intended purposes of this SQC QC
Entry Conditions which apply (tags) Penn Objectives INSPT'D [✓] EC
(Generic Entry Condition SI pg. 64–66)

Current Entry States (met, waived) Met Why? ______
Exit Conditions which will be applied (tags) Edit/CR/RemDefects [ ] XC
(Generic Exit Conditions SI pg. 202)

Meetings
Kickoff Date 3 Jul 2000 Location Room 4 Start Time 10.30 End time 11.00
Spec. Date 10 Jul 2000 Location Room 4 Start Time 10.00 End time 12.00
Process: Date 10 Jul 2000 Location Room 4 Start Time 12.15 End time 12.45

Documents (specified parts to be used by checkers)
samples or check(s)
Specification(s)
Penn Marketing Plan (all 4 pages)
Potts Marketing Plan (Pages 2,3)
Penn Objectives (Pages 2–5)

Rules: Generic [ ] SI 424-5 or in hours Rules.GR

Specific Rules.MP

Checklists: For Spec. Check.MP

For other Documents Check.OBJ

Team Setup:

Team Member Name	Tel. Ext.	SQC Role Soft. Insp. Page 362-73, e.g. Editor, Checker	Specification Part (The Specific section or pages of the document)	Source Documents and Sections you are responsible for	Rules & Checklists	Checking Procedure & other tactic	Checking Effort in hr.
Jenny Claire	2626	Checker	Penn Pages 2–3	Potts-MP	Rules.GR, Rules.MP, Check.MP	PCK PCC PCL PCB	2
Tom Franks	2533	Checker	Penn Pages 2–3	Potts-MP Section 2	Rules.MP Check.MP	PCK PCC PCL PCB	2
Harry Matthews	2522	Checker	Penn Page 1,4	Penn-OBJ Potts-MP	Rules.MP Check.MP	PCK PCC PSL PCB	2
Sam Murry	3321	Writer Checker, Editor	Penn Page 1,4	Penn-OBJ Potts-MP	Rules.GR Check.OBJ	PCK PCC POL PCB	2
Lucy Jones	3322	Team Leader	–	–	–	PLK PLC	–

Recommended Average Team Checking-Rates, SQC Goal and Strategy
Numeric SQC Goal, set during kind off ____________
Strategy to meet SQC Goal ____________

Optimum Checking Rate, for this type of specification is 1 pages per hour,
of non-commentary text.

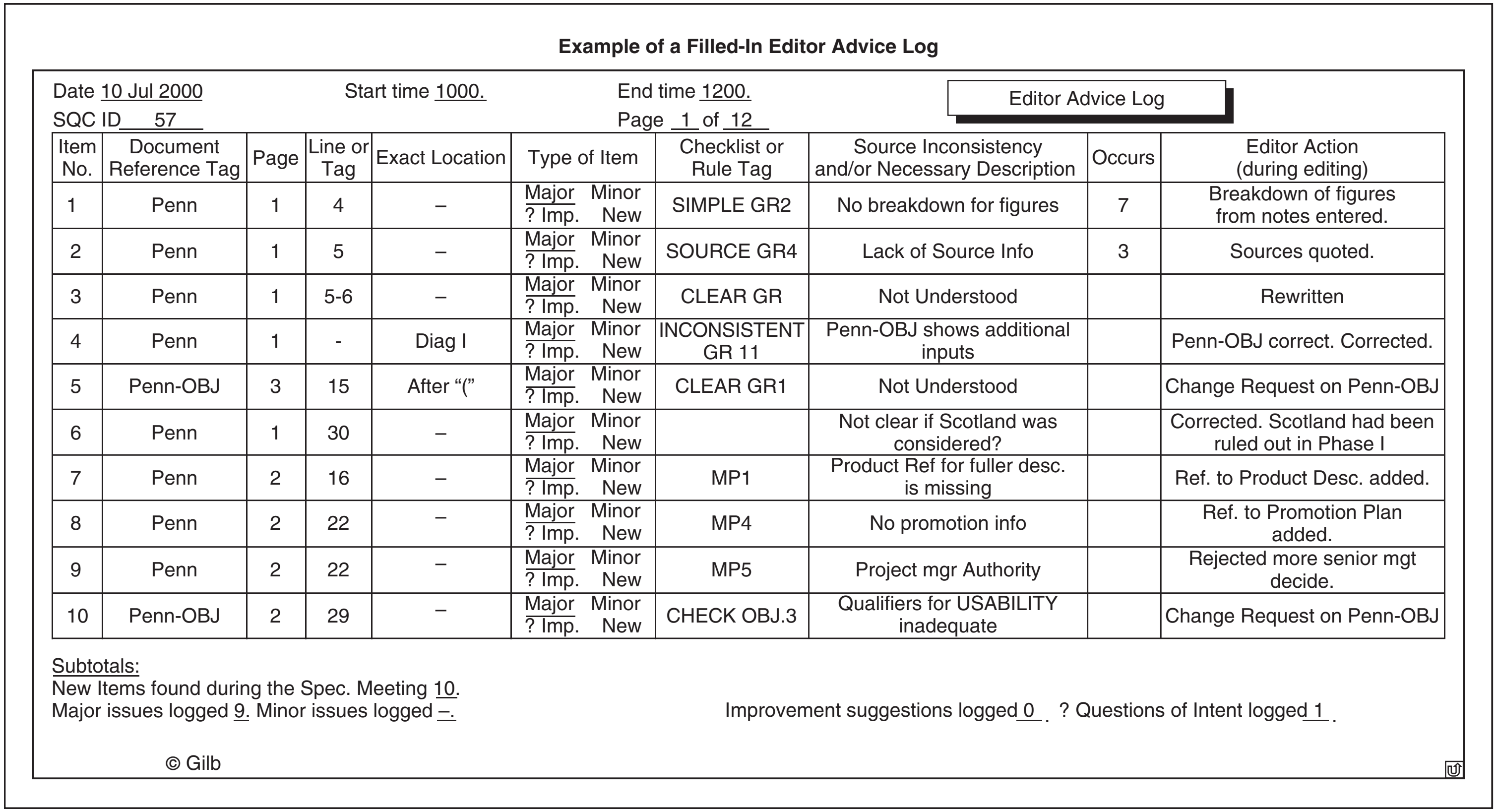
Spec meeting checking-rate: 2 page(s) (300 words, Non-Commentary) per hour
(optimum rate of checking during the Spec meeting)
Lucy Jones this is the end of the Master Plan.

© Gilb

____________ *Individual Checker Data Collection*
(filled in by each checker, after checking and before the Spec. meeting)

Actual work-hours (tenths) spent: ___ No. of (300w NC) Pages checked at optimum rate: ___
Major issues ___ [incl. Exxx-Majors (project threat) ___], minor issues ___
Process improvement suggestions ___ ?s of intent (to author) ___
My Checking Rate was: ___ Pages/hour.
How does this deviate from your planned rate? ______ Why? ______
____________ *and of Individual Data Collection*

Figure 8.2 Filled-in Master Plan.



Example of a Filled-In Editor Advice Log

Editor Advice Log

Date 10 Jul 2000 Start time 1000. End time 1200.

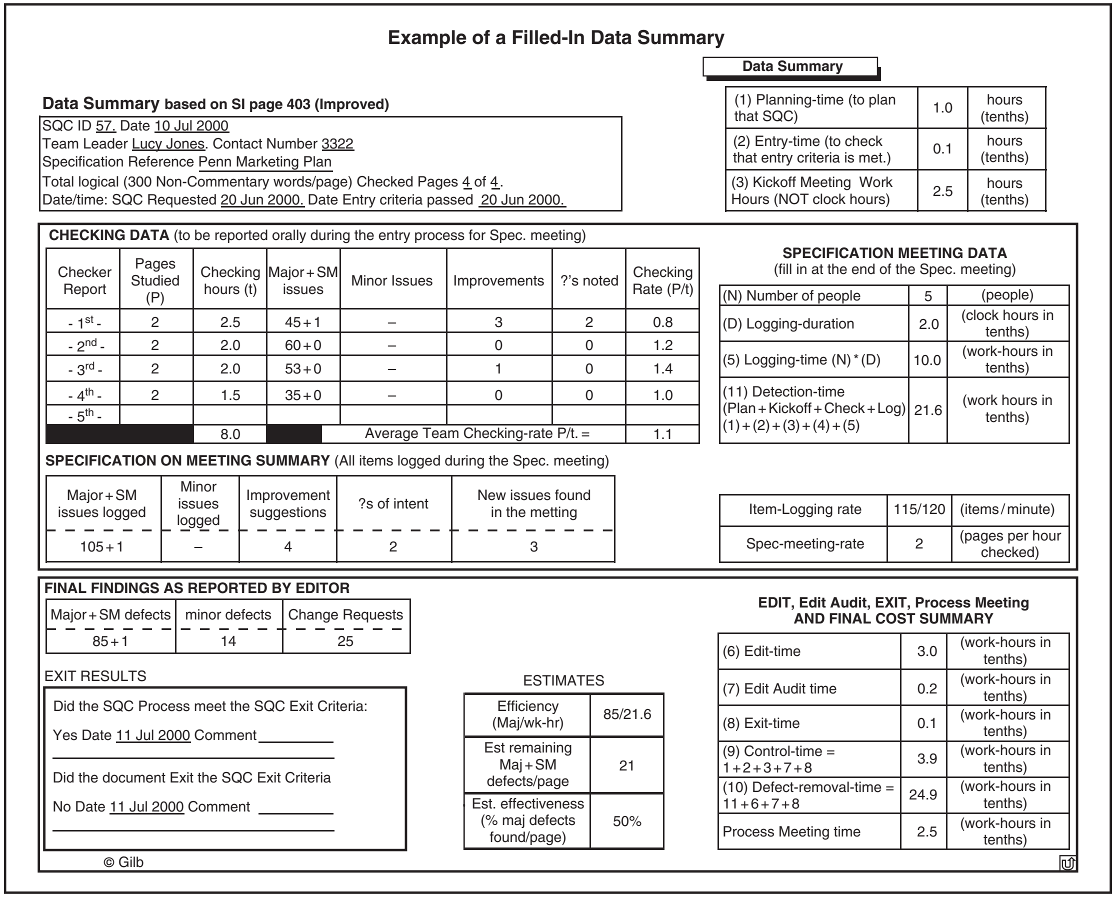
SQC ID 57 Page 1 of 12

Item No.	Document Reference Tag	Page	Line or Tag	Exact Location	Type of Item	Checklist or Rule Tag	Source Inconsistency and/or Necessary Description	Occurs	Editor Action (during editing)
1	Penn	1	4	–	Major Minor ? Imp. New	SIMPLE GR2	No breakdown for figures	7	Breakdown of figures from notes entered.
2	Penn	1	5	–	Major Minor ? Imp. New	SOURCE GR4	Lack of Source Info	3	Sources quoted.
3	Penn	1	5-6	–	Major Minor ? Imp. New	CLEAR GR	Not Understood		Rewritten
4	Penn	1	-	Diag I	Major Minor ? Imp. New	INCONSISTENT GR 11	Penn-OBJ shows additional inputs		Penn-OBJ correct. Corrected.
5	Penn-OBJ	3	15	After "("	Major Minor ? Imp. New	CLEAR GR1	Not Understood		Change Request on Penn-OBJ
6	Penn	1	30	–	Major Minor ? Imp. New		Not clear if Scotland was considered?		Corrected. Scotland had been ruled out in Phase I
7	Penn	2	16	–	Major Minor ? Imp. New	MP1	Product Ref for fuller desc. is missing		Ref. to Product Desc. added.
8	Penn	2	22	–	Major Minor ? Imp. New	MP4	No promotion info		Ref. to Promotion Plan added.
9	Penn	2	22	–	Major Minor ? Imp. New	MP5	Project mgr Authority		Rejected more senior mgt decide.
10	Penn-OBJ	2	29	–	Major Minor ? Imp. New	CHECK OBJ.3	Qualifiers for USABILITY inadequate		Change Request on Penn-OBJ

Subtotals:
New Items found during the Spec. Meeting 10.
Major issues logged 9. Minor issues logged –. Improvement suggestions logged 0 . ? Questions of Intent logged 1 .

© Gilb

Figure 8.3 Filled-in Editor Advice Log.



Example of a Filled-In Data Summary

Data Summary

(1) Planning-time (to plan that SQC)	1.0	hours (tenths)
(2) Entry-time (to check that entry criteria is met.)	0.1	hours (tenths)
(3) Kickoff Meeting Work Hours (NOT clock hours)	2.5	hours (tenths)

Data Summary based on SI page 403 (Improved)

SQC ID 57. Date 10 Jul 2000
Team Leader Lucy Jones. Contact Number 3322
Specification Reference Penn Marketing Plan
Total logical (300 Non-Commentary words/page) Checked Pages 4 of 4.
Date/time: SQC Requested 20 Jun 2000. Date Entry criteria passed 20 Jun 2000.

CHECKING DATA (to be reported orally during the entry process for Spec. meeting)

Checker Report	Pages Studied (P)	Checking hours (t)	Major + SM issues	Minor Issues	Improvements	?'s noted	Checking Rate (P/t)
- 1st -	2	2.5	45 + 1	–	3	2	0.8
- 2nd -	2	2.0	60 + 0	–	0	0	1.2
- 3rd -	2	2.0	53 + 0	–	1	0	1.4
- 4th -	2	1.5	35 + 0	–	0	0	1.0
- 5th -							
■	■	8.0	■	Average Team Checking-rate P/t. =			1.1

SPECIFICATION MEETING DATA
(fill in at the end of the Spec. meeting)

(N) Number of people	5	(people)
(D) Logging-duration	2.0	(clock hours in tenths)
(5) Logging-time (N) * (D)	10.0	(work-hours in tenths)
(11) Detection-time (Plan + Kickoff + Check + Log) (1) + (2) + (3) + (4) + (5)	21.6	(work hours in tenths)

SPECIFICATION ON MEETING SUMMARY (All items logged during the Spec. meeting)

Major + SM issues logged	Minor issues logged	Improvement suggestions	?s of intent	New issues found in the metting
105 + 1	–	4	2	3

Item-Logging rate	115/120	(items/minute)
Spec-meeting-rate	2	(pages per hour checked)

FINAL FINDINGS AS REPORTED BY EDITOR

Major + SM defects	minor defects	Change Requests
85 + 1	14	25

EXIT RESULTS

Did the SQC Process meet the SQC Exit Criteria:

Yes Date 11 Jul 2000 Comment __________

Did the document Exit the SQC Exit Criteria

No Date 11 Jul 2000 Comment __________

ESTIMATES

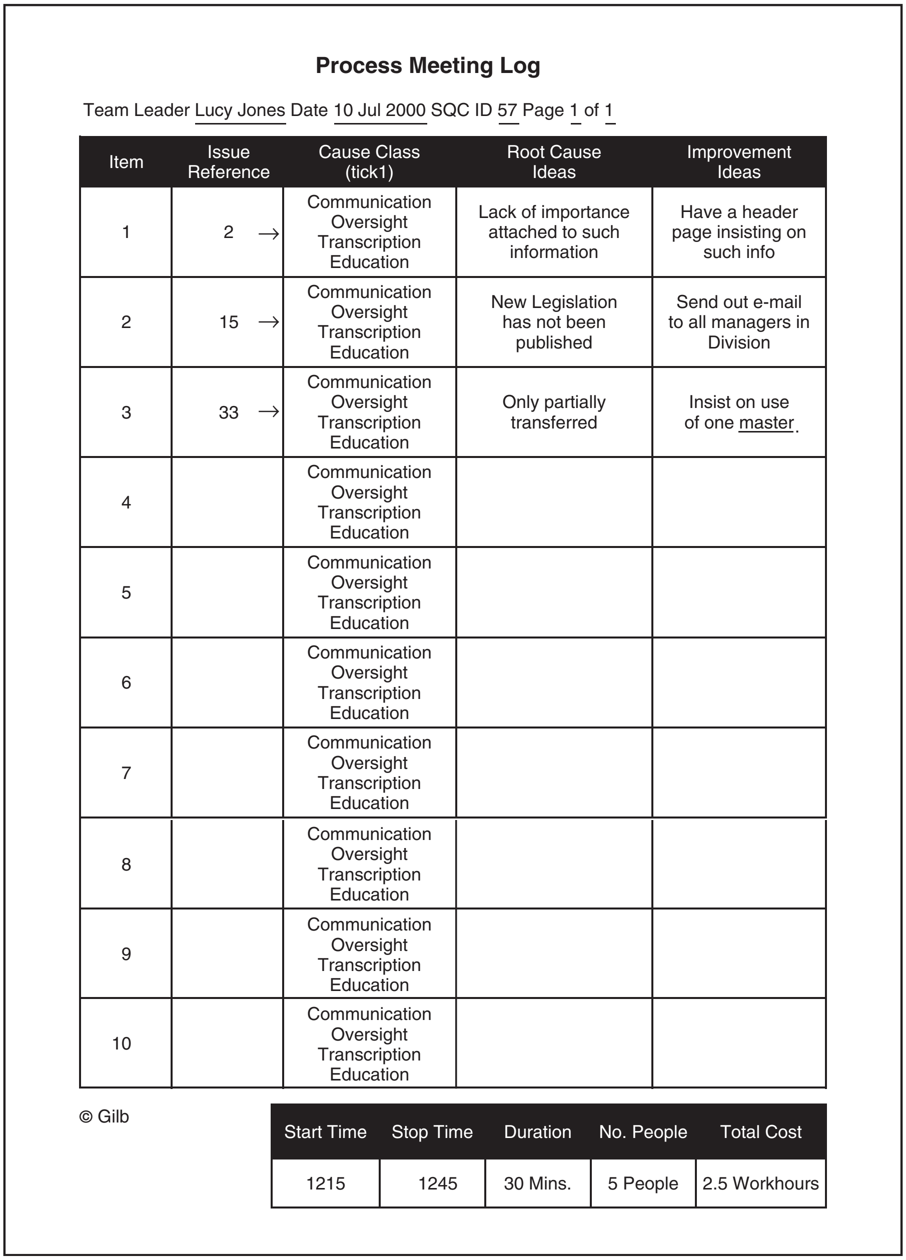
Efficiency (Maj/wk-hr)	85/21.6
Est remaining Maj + SM defects/page	21
Est. effectiveness (% maj defects found/page)	50%

EDIT, Edit Audit, EXIT, Process Meeting AND FINAL COST SUMMARY

(6) Edit-time	3.0	(work-hours in tenths)
(7) Edit Audit time	0.2	(work-hours in tenths)
(8) Exit-time	0.1	(work-hours in tenths)
(9) Control-time = 1 + 2 + 3 + 7 + 8	3.9	(work-hours in tenths)
(10) Defect-removal-time = 11 + 6 + 7 + 8	24.9	(work-hours in tenths)
Process Meeting time	2.5	(work-hours in tenths)

© Gilb

Figure 8.4 Filled-in Data Summary.



Process Meeting Log

Team Leader Lucy Jones Date 10 Jul 2000 SQC ID 57 Page 1 of 1

Item	Issue Reference	Cause Class (tick1)	Root Cause Ideas	Improvement Ideas
1	2 →	Communication Oversight Transcription Education	Lack of importance attached to such information	Have a header page insisting on such info
2	15 →	Communication Oversight Transcription Education	New Legislation has not been published	Send out e-mail to all managers in Division
3	33 →	Communication Oversight Transcription Education	Only partially transferred	Insist on use of one master
4		Communication Oversight Transcription Education		
5		Communication Oversight Transcription Education		
6		Communication Oversight Transcription Education		
7		Communication Oversight Transcription Education		
8		Communication Oversight Transcription Education		
9		Communication Oversight Transcription Education		
10		Communication Oversight Transcription Education		

© Gilb

Start Time	Stop Time	Duration	No. People	Total Cost
1215	1245	30 Mins.	5 People	2.5 Workhours

Figure 8.5
Filled-in Process Meeting Log.

SQC Process Roles and Responsibilities

An **efficient team** *(most major defects/work-hour) uses 2 or 3 people in total. An* **effective team** *(most major defects/page) uses a maximum of 3 to 5 people in total.*

Team Leader

A team leader is responsible for managing an SQC process. The team leader is responsible for knowing SQC thoroughly and helping the team members to perform. They follow the 'best-practice' SQC processes. An SQC team leader is normally trained for about a week, and is then formally approved to practice by virtue of their practical ability and continued correct practice. Inadequate SQC team leader training leads to failure of the SQC process (Grady and Van Slack 1994).[4]

Checker

Checkers are primarily 'consultants to the writer' and their detailed knowledge of the defectiveness of the writer's work is confidential. Almost all engineering team members work as checkers on occasion, including the writer and probably the team leader. (The team leader might choose to be a 'non-playing captain' of the team. They would not check in order to focus their time on the team leader responsibility or because they were not technically capable in the specification 'language.')

Checkers are SQC team members who actively check a set of documents: the main specification, its source specifications, kin specifications, the rules, checklists and procedures. They focus on using the checklists and rules to find major defects. Exactly which documents a specific checker uses, and what they check for, is determined by the role or roles assigned to them by the team leader.

Checkers are also invited to submit specific comment on possible improvements to the process and the process standards (procedures, rules, entry conditions, exit conditions and forms). They will, hopefully, get some insights during their checking work (for example, about the need for better rules).

[4] Grady reported that HP failed to achieve results from 1983 to 1988 until they properly trained their team leaders on a week-long course (designed and held, as cited there, by this author). This article is reproduced in Wheeler, Brykczynski and Meeson (1996).

Writer: Also Known as Author

The writer is the person currently responsible for writing or updating a specification. The SQC process serves the writer primarily: in confidence. SQC serves the organization secondarily.

Editor

The editor is usually the same person as the writer. The editor is the person, who takes over the issues in the Editor Advice Log, decides (based on standards) what action is required and carries it out. Some issues will be defects and need fixing. Some issues will require clarification. Some issues will be rejected and others will require change requests to other documents to be raised. The Editor Advice Log can be updated with the editor's decisions.[5]

In extreme cases, but unfortunately all too frequently, the defect density found (for example 90 majors in a page) will effectively spell out the fact that 'burning' the work and completely rewriting it will be more cost-effective.

Scribe

The scribe writes up the Editor Advice Log or other team notes at an SQC meeting. This can be any one of the team members. By default, the team leader will scribe. 'Who scribes' is not a critical decision.

Others

In a larger organizational setting, there are 'players' outside the team who support the SQC process. These include quality management, SQC process champions, process owners (for both SQC processes and the work processes, for example, 'Requirement process owner'), senior SQC team leaders, SQC process trainers and engineering data analysts (perhaps specialized in SQC data statistics). When the SQC process is applied to perform specification content reviews, the participants will be senior staff expected to use judgment and to take responsibility for the consequences of their approval.

[5] There are many ways to report what editing action has been undertaken and any suitable method is fine.

8.5 Process Description: Specification Quality Control

Process: Specification Quality Control

Tag: Process.SQC.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Note: See (Gilb and Graham 1993) for more detail on the sub-processes. All sub-processes are DDP unless marked as DPP.

Entry Conditions

E1: The specification writer must have requested the SQC hoping to get help and exit validation for the specification.

E2: A team leader for the SQC is found from amongst the '*approved* team leaders' group.

E3: All relevant documents (main specification, kin documents, source documents, rules, checklists and forms) are available and ideally have successfully exited SQC – apart from the main specification!

SQC Sub-Processes

Entry

The team leader ensures that the SQC *entry conditions* are met. This includes obtaining the relevant documents and checking their status. Entry conditions are evaluated *during* the Planning phase.

Planning

The team leader produces the master plan for the SQC (about 1 hour's work). This involves deciding what material within the specification is to be sampled, what documents are to be included, what rules must be used, who is going to be on the team and what their roles are. The optimum checking rate is determined based on history.

SQC Strategy

The team leader decides the purpose(s) of this SQC and ensures a suitable overall SQC strategy. Again this is evaluated *during* Planning.

Kickoff

The team meets at a Kickoff meeting, where the team leader makes sure every member knows what they *need to know* about the SQC process and the project documents, and that they are committed to working the plan *as a team*. The team may approve a suggested 'quantified goal' and 'appropriate strategy' to meet it (a DPP component).

Checking

The team members individually carry out their assigned defect-search roles at their self-adjusted optimum checking-rate, looking for major defects. They collect data about the cost and result of their personal checking activity. This process will typically, for a sample of about two logical pages, take two hours for each person. Checkers will ask the team leader for help if necessary. They will also report to the team leader any unusual or serious problems they discover that might impact the future course of the SQC process, for example, that the number of issues (potential defects) discovered is sufficiently large to consider abandoning the SQC.

Specification Meeting

This is a team meeting (real or virtual) of up to two hours duration. The duration and meeting content depend on data collected from Checking.

- *Checkers' Report*: At the beginning of the specification meeting, checkers *report* their data from Checking. The team leader evaluates this data and makes decisions about how the meeting and the rest of this SQC process should proceed. The meeting may be cancelled or modified in content and duration.
- *Issue Logging*: The checkers report their issues, mainly potential majors, which can be in any of the participating documents. A scribe logs issues in the Editor Advice Log. There should be no discussion concerning the issues discovered, just unconditional logging of the issue (the rule violation and its location in the specification). Checkers may also make process improvement suggestions (*Note: This is part of the DPP process*), and log technical 'questions of intent' to the writer. Issue logging within a specification meeting takes up to 30 minutes.
- *Double Checking*: If it is desired that additional defects are found, then double checking at the experienced specification meeting[6] optimum checking rate will be carried out during the meeting. This identifies

[6] This rate is similar but may vary by about 30% from the optimum rate average found for individual checking activity. In addition, it is a group activity rate and is not directly tunable to single individuals. Of course, single individuals will exploit the given time more or less effectively, depending on their personal ability and motivation.

about 15% additional major defects and adds about 1.5 hours to the meeting. This extra checking is only useful in 'cleanup mode,' not when sampling and measuring to determine exit (normal mode).

Process Meeting

After the specification meeting and a short break, the team optionally may spend up to 30 minutes, analyzing up to 10 logged potential major defects. For each chosen potential defect, one minute is spent describing the issue, one minute is spent brainstorming possible root causes and one minute is spent brainstorming preventative cures. This data will later be recorded in a quality assurance (QA) database as inputs ('grass root insights': suggestions, hints, ideas) to the organization's more-systematic and formal process improvement specialists. (*Note: A Process Meeting is part of the DPP process.*)

Edit

The editor (usually the specification writer) takes over the 'Editor Advice Log', which consists of the issues (that could warrant correction or action) logged at the specification meeting. The editor examines the logged issues, determines how to resolve them and then at least fixes the issues considered to be major defects. The editor may discover additional defects and should make corrections to any majors identified outside the sample checked. Other reasonable action is taken, such as sending out change requests to owners of other documents. An extreme edit is a full rewrite according to all rules.

Edit Audit

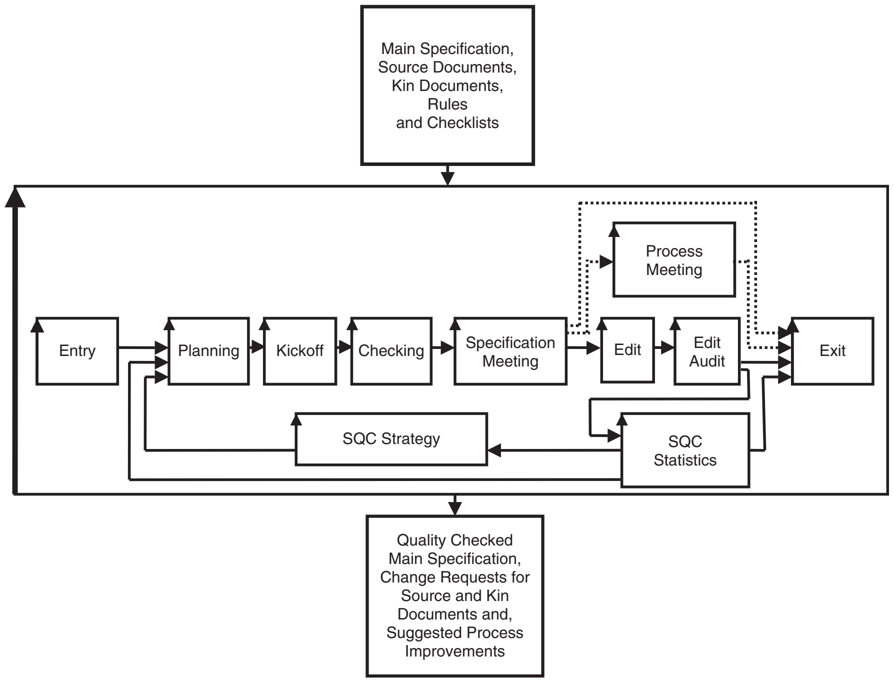
A process carried out by the team leader to verify that a reasonable and complete editing job has been done. Consequently, the editor takes formal responsibility for the editing. This can be done in minutes.

SQC Statistics

The team leader will ensure that all the required statistics from the SQC are captured in the SQC database. This assumes a process control use of SQC data.

Exit

The team leader evaluates the formal SQC exit conditions to see if the specification may be released 'economically' for normal use. The


Main Specification, Source Documents, Kin Documents, Rules and Checklists
Process Meeting
Entry
Planning
Kickoff
Checking
Specification Meeting
Edit
Edit Audit
Exit
SQC Strategy
SQC Statistics
Quality Checked Main Specification, Change Requests for Source and Kin Documents and, Suggested Process Improvements

Note: The 'Process Meeting' sub-process is exclusively a part of Defect Prevention Process (DPP). All the rest is Defect Detection Process (DDP), although there may be a small component of DPP within some of these sub-processes.

Figure 8.6
Diagram of the SQC Process showing the sub-processes.

estimated number of remaining major defects in the specification is especially important. If the main specification is not released, the team leader must work towards acceptable exit-levels of quality, usually in cooperation with the specification writer.

Exit Conditions

X1: The main specification must have fewer remaining major defects/ page than the agreed exit standard (a maximum of 'one remaining' is a reasonable ambition level, initially).

A Simplified SQC Process

SQC as described in the procedure above is the full-scale version. There are situations when a simplified SQC process is more appropriate (for example, to obtain a rapid assessment of the specification quality of a contract or to demonstrate to management some of the power of SQC to get their 'buy-in').

A ‘Simplified SQC Process’ is presented below.

Note: There are several limitations to this simplified process:

- it is only a small sample so the accuracy is not as good as a full or larger sample
- the team will not have time or experience to get up to speed on the rules and the concept of major defect
- a small team of two people does not have the known effectiveness of three or four people
- you will not have the basis for making corrections to the entire specification
- the checking will not have been carried out against all the possible source documents. (Usually in the simplified SQC process, no source documents are used and memory is relied on. While this means that the checking is not nearly as accurate, it does considerably speed up the process.)

However, if the sample turns up a defects density estimation of 50 to 150 major defects/page (which is quite normal), that is more than sufficient to convince the people participating, and their managers, that they have a serious problem.

The immediate solution to the problem of high defect density is not to remove the defects from the document. The most effective practical solution is to make sure each individual specification writer takes the defect density criteria (and its ‘no exit’ consequence) seriously. They will then learn to follow the rules and, as a result, will reduce their personal defect injection rate. On average, a personal defect injection rate should fall by about 50% after each experience of using the SQC process. Widespread use of SQC will result in large numbers of engineers learning to follow the rules.

To get to the next level of quality improvement, the next step is to improve the rules themselves.

Simplified SQC Process

Tag: Simplified SQC.
Version: October 7, 2004.
Owner: Tom@Gilb.com.
Status: Draft.

Entry Conditions

- A group of two, or more, suitable people* to carry out Simplified SQC is assembled in a meeting.
- These people have sufficient time to complete a Simplified SQC. Total elapsed time: 30 to 60 minutes.
- There is a trained SQC team leader at the meeting to manage the process.

Procedure

P1: Identify Checkers: Two people, maybe more, should be identified to carry out the checking.
P2: Select Rules: The group identifies about three rules to use for checking the specification. (My favorites are clarity ('clear enough to test'), unambiguous ('to the intended readership') and completeness ('compared to sources'). For requirements, I also use 'no design'.)
P3: Choose Sample(s): The group then selects sample(s) of about one page in length (300 non-commentary words). Choosing a page at random can add credibility – so long as it is representative of the content subject to quality control. The group should decide whether all the checkers should use the same sample or whether different samples are more appropriate.
P4: Instruct Checkers: The SQC team leader briefly instructs the checkers about the rules, the checking rate, and how to document any issues and determine if they are major defects (majors).
P5: Check Sample: The checkers use between 10 and 30 minutes to check their sample against the selected rules. Each checker should 'mark up' their copy of the document as they check (underlining issues and classifying them as 'major' or not). At the end of checking, each checker should count the number of 'possible majors' they have found in their page.
P6: Report Results: The checkers each report to the group their number of 'possible majors.' The SQC team leader leads a discussion to determine how many of the 'possible majors' are actually likely to be majors. Each checker determines their number of majors and reports it.
P7: Analyze Results: The SQC team leader extrapolates from the findings the number of majors in a single page (about 6 times** the most majors found by a single person, or alternatively 3 times the unique majors found by a 2 to 4 person team). This gives the major defect density. If using more than one sample, average the densities found by the group in different pages. The SQC team leader then multiplies this average major defects/page density by the total number of pages to get the total number of major defects in the specification (for dramatic effect!).
P8: Decide Action: If the number of majors/page found is a large one (ten majors or more), then there is little point in the group doing anything, except determining how they are going to get someone to write the specification properly. There is no economic point in looking at the other pages to find 'all the defects', or correcting the majors already found. There are too many majors not found.
P9: Suggest Cause: Choose any major defect and think for a minute why it happened. Then give a short sentence, or better still a few words, to capture your verdict.

Exit Conditions

- Exit if less than 5 majors/page extrapolated total density, or if an action plan to 'rewrite' has been agreed.

Notes:

* A suitable person is anyone, who can correctly interpret the rules and the concept of 'major.'
** Concerning the factor of multiplying by '6': We have found by experience (Gilb and Graham 1993: Bernard) that the total unique defects found by a team is approximately twice that of the number found by the person who finds the most defects in the team. We also find that inexperienced teams using Simplified SQC seem to have about one third effectiveness in identifying the major defects that are actually there. So $2 \times 3 = 6$ is the factor we use (or $3 \times$ the number of unique majors found by the team).

Simplified Specification Quality Control Form

SQC Date: **May 29, 200X**. SQC Start Time: ______
SQC Leader: **Tom**.
Author: **Tino**.
Other Checkers: **Artur**.

Specification Reference: **Test Plan**.
Specification Date and/or Version: **V 2** Total Physical Pages: **10**.

Sample Reference within Specification: **Page 3**.
Sample Size (Non commentary words): **approx. 300**.

Rules used for Checking: **Generic Rules, Test Plan Rules.**
Planned Exit Level (Majors/logical page): ______ or less.

Checking Time Planned (Minutes): **30**. Actual: **25**.
Checking Rate Planned (Non commentary pages/hour): **2**.
(*Note this rate should be less than 2 logical pages/hour*)

Actual Checking Rate (Non commentary words/minute): ______
Number of Defects Identified by each Checker:
Majors: **6, 8, 3**. Total Majors Identified in Sample: **17**.
Minors: **10, 15, 30**.

Estimated Unique Majors Found by Team: **16 ± 5**.
(*Note 2 × highest number of Majors found by an individual checker*)

Estimated Average Majors/Logical Page: ~**16** × **3** = **48**.
(*A Logical Page = 300 Non commentary words*)
Majors in Relation to Exit Level: **48/1 (47 too many)**.
Estimated Total Majors in entire Specification: **48** × **10** = **480**.
Recommendation for Specification (*Exit/Rework/Rewrite*): **No exit, redo and resubmit**.

Suggested Causes (of defect level): **Author not familiar with rules**.

Actions suggested to mitigate Causes: **Author studies rules, All authors given training in rules**.

Person responsible for Action: **Project Manager**.
SQC End Time: **18:08**. Total Time taken for SQC: ______

Version: August 15, 2004. Owner: Tom@Gilb.com

8.6 Principles: Specification Quality Control

1. **The Principle of 'Illegality'**
   'Defects' are *objective* violation of accepted written rules.
2. **The Principle of 'Majors are the pay off'**
   Major defects are the only *economically interesting* defects.
3. **The Principle of 'Keen to be seen clean'**
   The main purpose of SQC is to measure that the specification is clean enough: *not* to clean up a specification that *isn't.*
4. **The Principle of 'Cleanup your own mess'**
   Specification cleanup is the writer's responsibility, *before* SQC.
5. **The Principle of 'Prevention is better than cure'**
   There are many effects of SQC, but the most useful are learning to avoid defects caused by bad process, and committed by the writer.
6. **The Principle of '50% effectiveness'**
   History shows that you can only expect to find and fix about *half* the defects that are there.
7. **John Craven's Principle (within Hewlett Packard)**
   The team is there to make the "writer look like a *hero.*"
8. **The Principle of 'Magnificent Profitability'**
   The expected return on investment for SQC is at least 'ten to one.'
9. **The Principle of 'Client-Server'**
   The writer is the *client* and the checkers serve as advisors.
10. **The Principle of 'The Pilot in Command'**
    The team leader is responsible for the SQC process.

Good execution of a badly executed specification will tend to execute you!

8.7 Additional Ideas: Specification Quality Control

There are some *central* ideas of SQC, which are worth looking at in more depth:

Economics of using SQC

The cost of finding and fixing defects has to be balanced against the benefit of removing the defects. The cost of fixing a defect escalates the longer it is left unfixed. In general, as we move from requirements/

design-stages to test-stages, the total system-wide costs of *removing* major defects increase by an order-of-magnitude. As we move into design implementation, manufacturing, installation, servicing and distribution, yet *another* additional order-of-magnitude of cost is generally our penalty for dealing with major defects later.

The cost of finding defects also varies. There are several QC nets that specifications pass through as a product is developed. Also, just because there *is* a defect doesn't mean that it *will* cause damage. This is where sampling and understanding your document quality level is essential.

If there is more than approximately one remaining major defect/page, it will tend to pay off to fight the defects immediately, using SQC, rather than downstream. With less than that, it probably pays off to let that major defect (exact location unknown) slip through *this particular* QC net, and hope it is still caught in some other QC net in the systems engineering process.

Unfortunately most real engineering environment 'approved' documents are at least one order-of-magnitude *worse* quality than one major defect/page: 10 to 20 or far more major defects/page is common, according to my frequent measurements. But without SQC, to measure for us, we don't 'know' this.

Effectiveness of SQC

If SQC is consistently carried out according to official guidelines (critically including the 'checking rate' being at the optimum level), then experience in IBM Rochester Labs, MN (Gilb and Graham 1993 Page 23) shows that the defect-finding *effectiveness is relatively stable.* Thirty percent effectiveness is a beginner's level (my experience). For a mature SQC process, effectiveness, for a single-pass attempt, tends to be in the range 60% to 90% (Gilb and Graham 1993: IBM Experience), depending on the type of specification being checked. By systematic SQC process improvement, the effectiveness can slowly be improved to its maximum potential. Cumulative SQC effectiveness, for multiple passes, has been shown to reach a maximum of 95% (IBM UK and Sema UK Case (Gilb and Graham 1993: Leigh, D.)).

Determining Effectiveness and Estimating Remaining Defects

SQC can be used directly to *measure* defects found and, indirectly to *estimate* the defects not found. Providing that 'effectiveness' (% of 100%) at finding defects is known and is relatively stable, it can be used to estimate the number of *unfound* defects. Effectiveness (of your teams on your specifications at finding defects) can be determined in two ways:

1. For a specific specification, we can measure the percentage of the 'available' defects, which a given SQC process was able to find. We can work out this percentage if we know the number of defects found by the SQC and the total number of defects found at later stages by other QC processes, testing and field use. IBM has practiced this for decades in software engineering (Kaplan, Clark and Tang 1994; IBMSJ 1994).
2. Another method, faster and cheaper, and more credible locally, is to repeat SQC on the same sample. If we find 30 defects on first attempt, fix them all, and hypothesize that we are 30% effective at finding them, this means we should have about 70 defects not found in our sample, right? If, after fixing all the 30 we found, the second SQC, done on the same sample, consistently finds about 21 defects (± about 6), it would confirm our prediction of 30% (21 of the 70 remaining from the first SQC). The '% of available defects found' is the effectiveness of the given SQC process. We use this method regularly on our training courses, and it works! It will also work for *any* test or QC process.

Once you have determined your effectiveness, you can estimate the remaining defects in a specification. We use the number of 'estimated probable remaining defects' to decide if a specification can exit (a typical exit condition is 'no more than one probable remaining major defect/ page'). See Figure 8.7 for the calculation of 'estimated probable remaining defects.' We use effectiveness to determine the number of defects unidentified, and then we improve the accuracy by considering the effects of the editing. One sixth of fix attempts during editing fail (M. Fagan 1986:[7] IBM experience), unless an SQC for each fix is done to reduce fix failure (IBMSJ 1994: Kan). In addition, defect injection occurs during editing as a side effect of the fixes; the defect injection rate is sometimes 2% to 5% – but is highly variable and uncertain.

The final consideration is the uncertainty in the estimate. I have found that this remaining defect estimate is reasonably correct, and even in poor circumstances is ±30% uncertain, which is good enough for most purposes.

A specification can have 'too high a density' of major defects (equals serious engineering-cost rule violations) to be acceptable for use (to be allowed 'entry' or 'exit'). *'We will find it in test'* is a dangerous delay. Delaying *action* on your specification's major defects threatens not only cost (thus profit), but time-to-market and competitive quality. It pays off to deal with most major defects *early*.

[7] Fagan. M. E, 'Advances in Software Inspections', *IEEE Transactions on Software Engineering*. Volume SE-12, Number 7, Pages 744–751, July 1986.

Estimating the Remaining Major Defect Density

Assumptions:
A logical page (page) is 300 non-commentary words.

- 30 major defects/page have been found during SQC.
- Your SQC effectiveness is 60% and your SQC is a statistically stable process.
- One sixth of your attempts to fix defects fail (One sixth is average failure to fix).
- New defects are injected during your attempts to fix defects at 5%.
- The uncertainty factor in the estimation of remaining defects is $\pm$30%.

Probable remaining major defects/page = 'Probable unidentified majors' + 'Bad fix majors' + 'Majors injected'

Let E = Effectiveness expressed as a percentage (%) = 60%

Probable unidentified majors = major defects acknowledged-by-editor for each page at Edit $\times$ (100 − E) / E = 30 major defects/page found $\times$ (100 − 60) / 60 = **20 major defects/page**.

Bad fix majors = One sixth of fixed majors = So, of 30 attempted fixes, **5 major defects in each page** are not fixed.

Majors injected = 5% of majors attempted to be fixed = **1.5 major defects/page**.

Probable remaining major defects/page = 20 + 5 + 1.5 = 26.5 remaining major defects/page.

Taking into account the uncertainty factor of $\pm$30% and rounding down to the nearest whole number gives **26 $\pm$ 7 Remaining Major Defects/Page**

(Minimum = 19, Maximum = 33 remaining major defects/page).

Figure 8.7
Estimating Remaining Major Defect Density: the main specification exit condition.

SQC and Rules

SQC is completely dependent on the rules that are applied. Just because you exit from an SQC process does not mean that all quality checking has been completed. It simply means that checking has been completed against the rules actually used, and has demonstrated an acceptable defect level.

By using different rules, different types of quality checking can be achieved. It is not simply a case of using the relevant rule set to match the type of specification. You need to consider what type of defects you are checking for and their potential cost if not detected.

Extending from SQC into Specification Review

There is no reason why the SQC method shouldn't be used to prepare for management reviews. You might have checked the content of a specification for consistency, completeness and clarity (Specification Rules).[8] But maybe you have not yet checked for the relevance to

[8] Note: This chapter mainly discusses and illustrates SQC from the viewpoint of checking for specification clarity, completeness and consistency. This ties in with the rules found in the other chapters, which are *Specification Rules*.

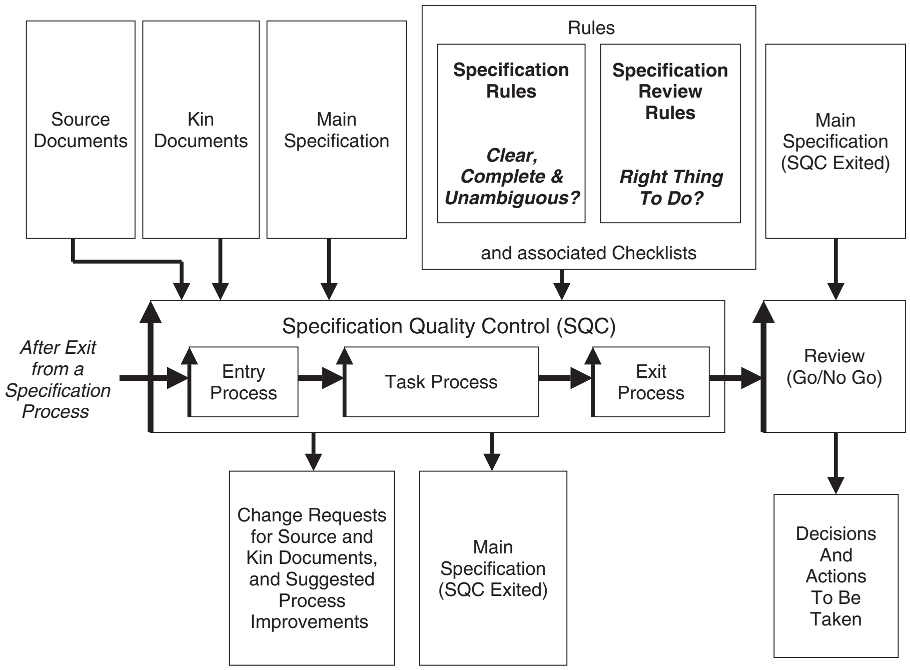

Source Documents
Kin Documents
Main Specification
Rules
Specification Rules
Clear, Complete & Unambiguous?
Specification Review Rules
Right Thing To Do?
and associated Checklists
Main Specification (SQC Exited)
Specification Quality Control (SQC)
After Exit from a Specification Process
Entry Process
Task Process
Exit Process
Review (Go/No Go)
Change Requests for Source and Kin Documents, and Suggested Process Improvements
Main Specification (SQC Exited)
Decisions And Actions To Be Taken

Figure 8.8
Overview of the SQC process showing how Specification Review Rules fit alongside Specification Rules.

current business or technical demands. For example, maybe a certain level of ROI is demanded? Maybe a specific safety margin must be shown to be present? By creating a different set of rules, called Specification Review Rules, the SQC process can also be used to carry out pre-review quality control, in advance of a review meeting. This will probably result in better quality control than would be carried out in a hurried executive review meeting.

8.8 Further Example/Case Study: A Stitch in Time Saves Nine

Trevor Reeve made use of SQC (at that time he called it 'Statistical Quality Control applied to Software and Documentation' or 'Documentation Quality Improvement') in all industrial aspects of a 1,500 person defense electronics manufacturer (later a part of Racal). He documented four years of experience after this author ran a course on-site (Gilb and Graham 1993, Pages 305–316).

Reeve carried out an analysis of the first 1,000 major defects logged by the SQC process to investigate the cost savings of using SQC. Test

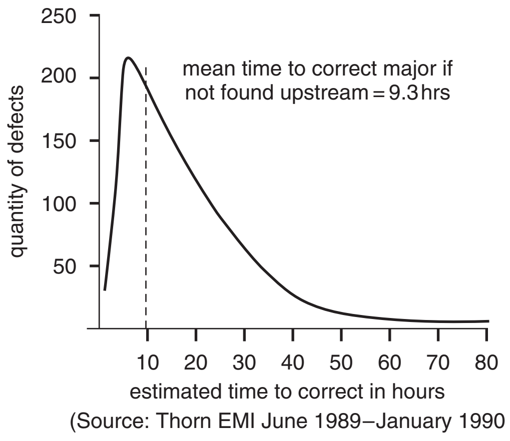


Figure 8.9
MEL/Thorn EMI (later RACAL) UK, Factory and Lab-wide SQC gave order-of-magnitude savings. About 1,000 major defects found by using SQC with multi-disciplinary teams were analyzed. The alternative cost to fix majors, if caught downstream, was nine times greater than if caught upstream by SQC. This is a frequency chart for the 1,000 defects.

and field staff were asked when these defects would have been found in their test or field reports. They were also asked to indicate what the cost of finding and fixing them would be. The frequency curve in Figure 8.9 was drawn based on their answers. The mean time to correct these defects downstream would have been 9.3 hours. The mean time to find and fix them using SQC was about one hour. The defects would *otherwise* have been found by test and by customers. The result of this was that it was acknowledged by top management that removing a major defect using SQC gave a net saving of about 8 hours, or a 9.3 to 1 ratio of engineering hours 'return' on investment in SQC.

Compare this to the Raytheon return of 7.7 to 1 (*see Section 1.8*). Six hundred inspections had been done at Thorn EMI by 1992, and over a thousand by early 1993. Savings were conservatively estimated at £500,000 each year, after one-time startup costs of £50,000. External consultants are said to have estimated real savings at double this figure. "Quality increased and development time has been reduced significantly."

Use of SQC on many different types of document

SQC experience at Raytheon was limited to software, but at MEL/ Thorn EMI, the documents

> "ranged from system, hardware and software design documents to software code and change notes . . . test specifications, proposals, program management documents (for example,

> configuration and program plans), contracts and purchase specifications, printed circuit board design and test specifications, and procedures and standards.
>
> Further application (1993) all contractual documents, drawings and internal specifications (for example, information technology and financial requirements) . . . with all contracts using it to a lesser or greater degree by end 1992. . . .
>
> The product appraisal process was revised to incorporate the technique into the normal activities performed by the organization on all types of document from contract to lowest level design and test specification including drawings." (Gilb and Graham 1993: P310)

Note: The use of SQC on the upstream documents will produce the greatest benefits because defects will be caught earlier, and do less damage.

The Organization Must be Supportive and SQC Needs a Champion

Since 1993, Trevor has confirmed many times within different organizations and various parts of the world, that two of the main factors for SQC to succeed are as follows:

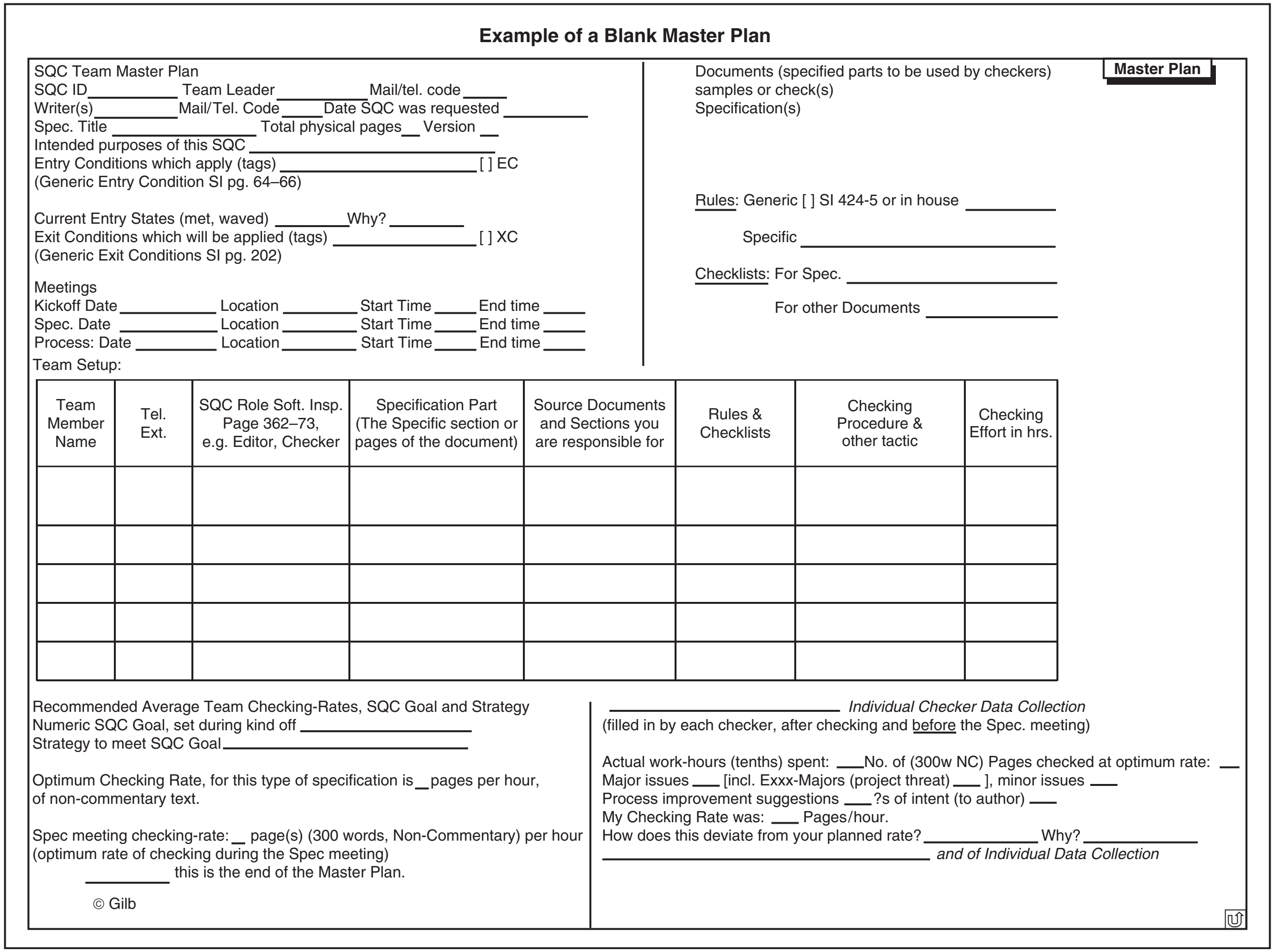
1. An organization really needs to be willing to change, and
2. The continuous presence of a totally committed champion of the method is necessary, for many years after the initial introduction of the method, to help the necessary culture change to take place. (This was also the experience in the same period of another client, Hewlett-Packard (Grady and Van Slack 1994).)

8.9 Diagrams/Icons: Specification Quality Control

This section shows the SQC forms as follows:

- Figure 8.10: Master Plan
- Figure 8.11: Editor Advice Log
- Figure 8.12: Data Summary
- Figure 8.13: Process Meeting Log
- Figure 8.14: Simplified SQC Form

These are the blank versions of the filled-in forms given earlier in this chapter.



Example of a Blank Master Plan

Master Plan

SQC Team Master Plan
SQC ID__________ Team Leader__________ Mail/tel. code_____
Writer(s)__________Mail/Tel. Code_____Date SQC was requested __________
Spec. Title _________________ Total physical pages__ Version __
Intended purposes of this SQC _____________________________
Entry Conditions which apply (tags) _______________________[ ] EC
(Generic Entry Condition SI pg. 64–66)

Current Entry States (met, waved) ________Why? ________
Exit Conditions which will be applied (tags) _________________[ ] XC
(Generic Exit Conditions SI pg. 202)

Meetings
Kickoff Date___________ Location ________Start Time_____End time _____
Spec. Date ___________ Location ________Start Time_____End time _____
Process: Date __________ Location ________Start Time_____End time _____

Documents (specified parts to be used by checkers)
samples or check(s)
Specification(s)

Rules: Generic [ ] SI 424-5 or in house __________

Specific ______________________________

Checklists: For Spec. ______________________

For other Documents ______________

Team Setup:

Team Member Name	Tel. Ext.	SQC Role Soft. Insp. Page 362–73, e.g. Editor, Checker	Specification Part (The Specific section or pages of the document)	Source Documents and Sections you are responsible for	Rules & Checklists	Checking Procedure & other tactic	Checking Effort in hrs.

Recommended Average Team Checking-Rates, SQC Goal and Strategy
Numeric SQC Goal, set during kind off ____________________
Strategy to meet SQC Goal_____________________________

Optimum Checking Rate, for this type of specification is__pages per hour,
of non-commentary text.

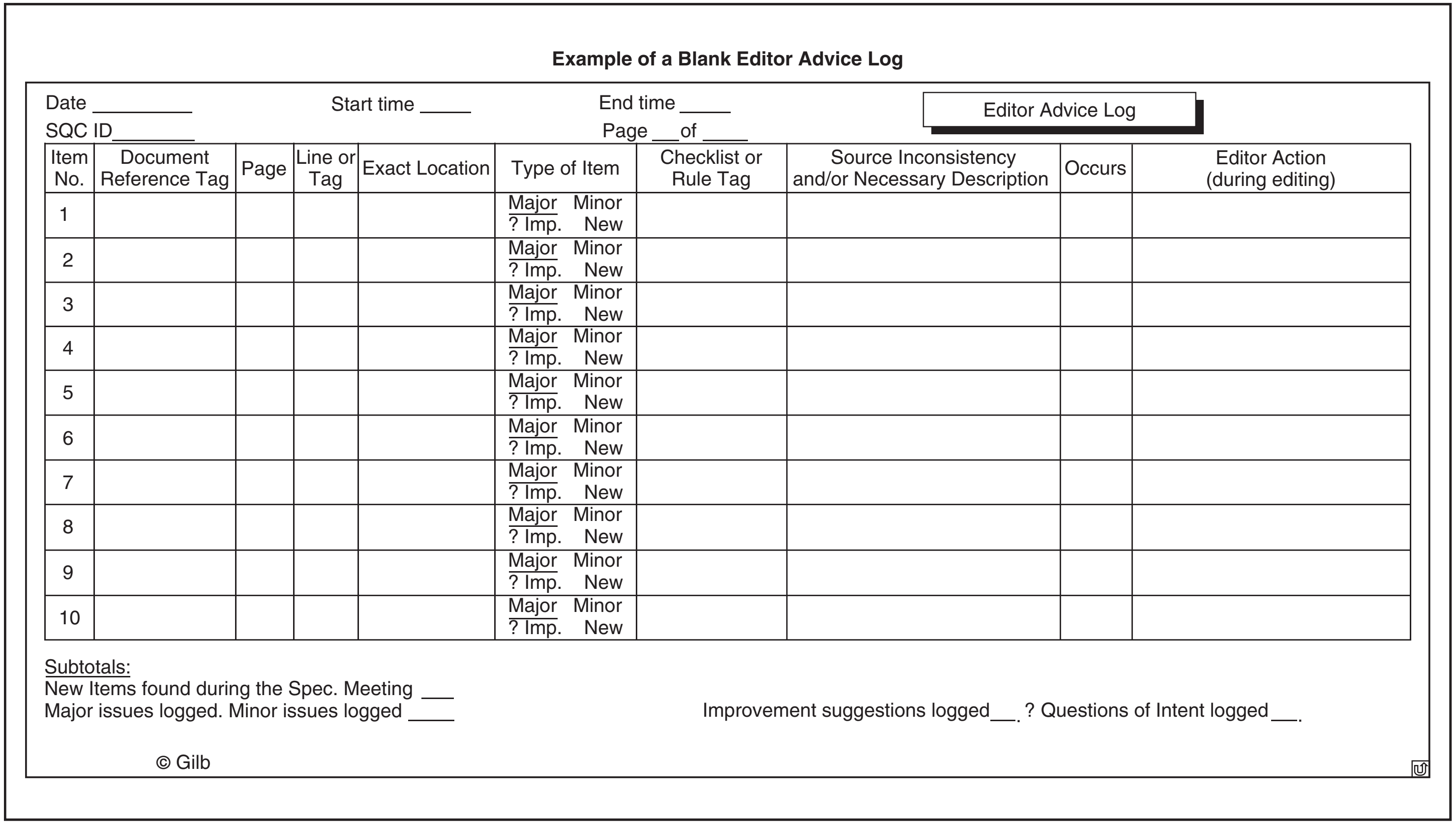
Spec meeting checking-rate:__ page(s) (300 words, Non-Commentary) per hour
(optimum rate of checking during the Spec meeting)
__________ this is the end of the Master Plan.

© Gilb

___________________________ *Individual Checker Data Collection*
(filled in by each checker, after checking and before the Spec. meeting)

Actual work-hours (tenths) spent: ___No. of (300w NC) Pages checked at optimum rate: __
Major issues ___ [incl. Exxx-Majors (project threat) ___], minor issues ___
Process improvement suggestions ___?s of intent (to author) ___
My Checking Rate was: ___ Pages/hour.
How does this deviate from your planned rate?_____________Why?_____________
_______________________________________ *and of Individual Data Collection*

Figure 8.10 Blank Master Plan.



Example of a Blank Editor Advice Log

Editor Advice Log

Date __________ Start time _____ End time _____

SQC ID________ Page ___of _____

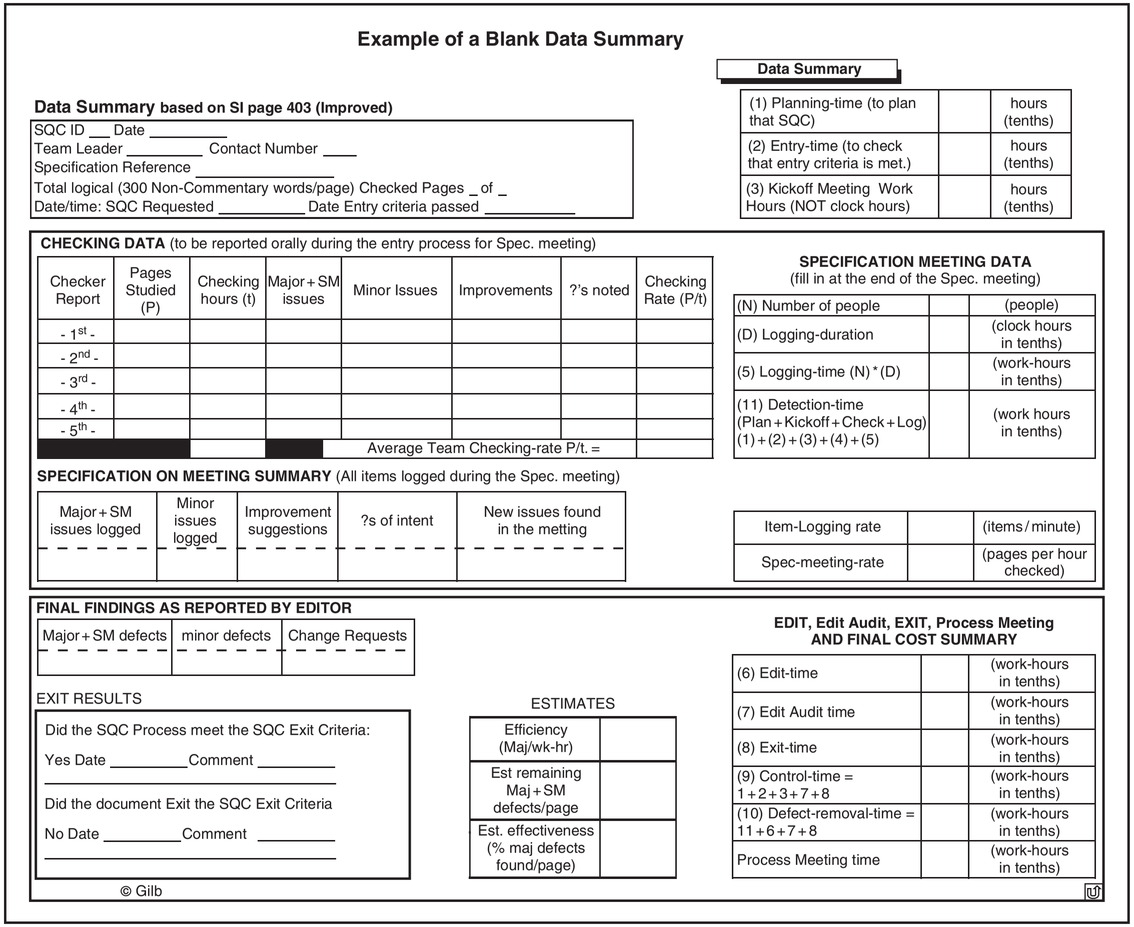
Item No.	Document Reference Tag	Page	Line or Tag	Exact Location	Type of Item	Checklist or Rule Tag	Source Inconsistency and/or Necessary Description	Occurs	Editor Action (during editing)
1					Major Minor ? Imp. New				
2					Major Minor ? Imp. New				
3					Major Minor ? Imp. New				
4					Major Minor ? Imp. New				
5					Major Minor ? Imp. New				
6					Major Minor ? Imp. New				
7					Major Minor ? Imp. New				
8					Major Minor ? Imp. New				
9					Major Minor ? Imp. New				
10					Major Minor ? Imp. New				

Subtotals:
New Items found during the Spec. Meeting ___
Major issues logged. Minor issues logged _____

Improvement suggestions logged___. ? Questions of Intent logged___.

© Gilb

Figure 8.11 Blank Editor Advice Log.



Example of a Blank Data Summary

Data Summary **based on SI page 403 (Improved)**

SQC ID ___ Date __________
Team Leader __________ Contact Number ____
Specification Reference __________________
Total logical (300 Non-Commentary words/page) Checked Pages _ of _
Date/time: SQC Requested ___________ Date Entry criteria passed ____________

Data Summary

(1) Planning-time (to plan that SQC)		hours (tenths)
(2) Entry-time (to check that entry criteria is met.)		hours (tenths)
(3) Kickoff Meeting Work Hours (NOT clock hours)		hours (tenths)

CHECKING DATA (to be reported orally during the entry process for Spec. meeting)

Checker Report	Pages Studied (P)	Checking hours (t)	Major+SM issues	Minor Issues	Improvements	?'s noted	Checking Rate (P/t)
- 1st -							
- 2nd -							
- 3rd -							
- 4th -							
- 5th -							
				Average Team Checking-rate P/t. =			

SPECIFICATION MEETING DATA
(fill in at the end of the Spec. meeting)

(N) Number of people		(people)
(D) Logging-duration		(clock hours in tenths)
(5) Logging-time (N) * (D)		(work-hours in tenths)
(11) Detection-time (Plan+Kickoff+Check+Log) (1)+(2)+(3)+(4)+(5)		(work hours in tenths)

SPECIFICATION ON MEETING SUMMARY (All items logged during the Spec. meeting)

Major+SM issues logged	Minor issues logged	Improvement suggestions	?s of intent	New issues found in the metting

Item-Logging rate		(items/minute)
Spec-meeting-rate		(pages per hour checked)

FINAL FINDINGS AS REPORTED BY EDITOR

Major+SM defects	minor defects	Change Requests

EXIT RESULTS

Did the SQC Process meet the SQC Exit Criteria:

Yes Date __________Comment __________
____

Did the document Exit the SQC Exit Criteria

No Date __________Comment __________
____

© Gilb

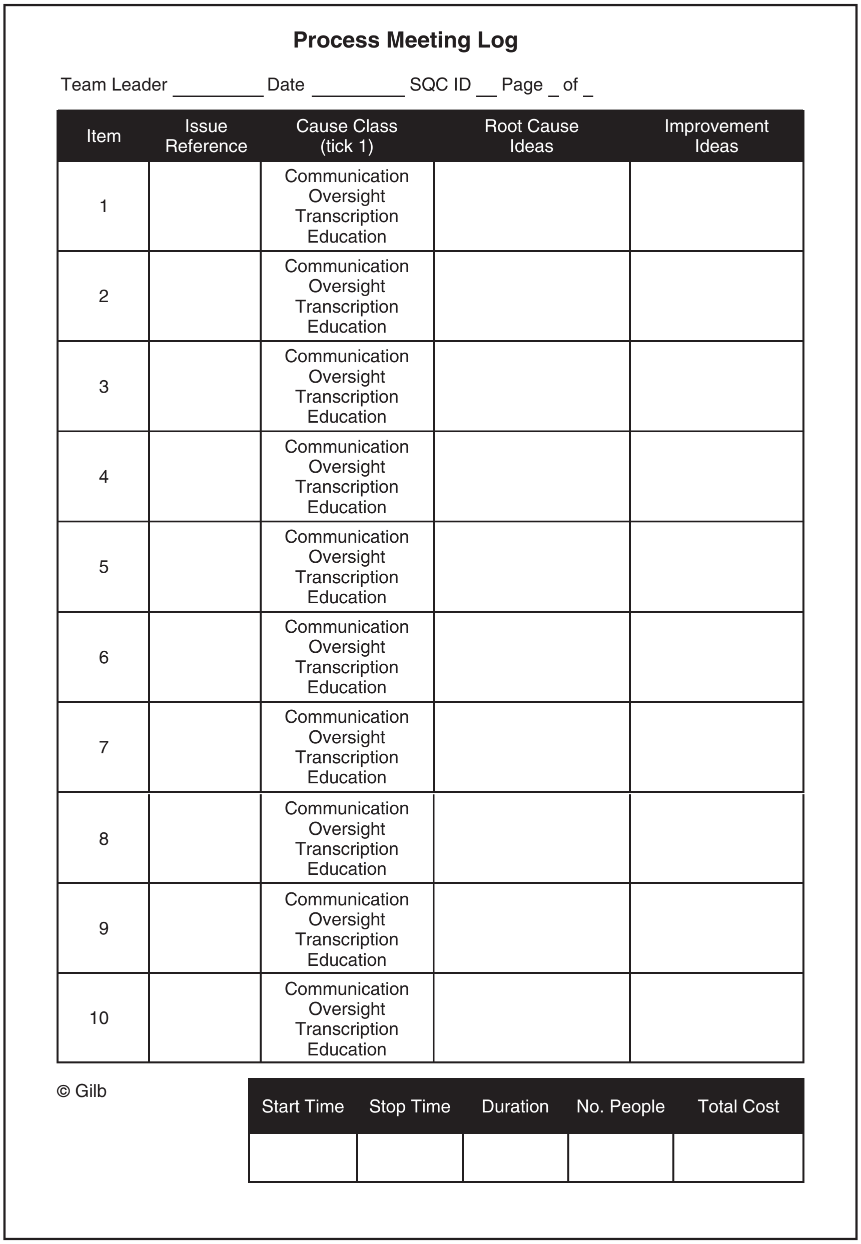
ESTIMATES

Efficiency (Maj/wk-hr)	
Est remaining Maj+SM defects/page	
Est. effectiveness (% maj defects found/page)	

EDIT, Edit Audit, EXIT, Process Meeting AND FINAL COST SUMMARY

(6) Edit-time		(work-hours in tenths)
(7) Edit Audit time		(work-hours in tenths)
(8) Exit-time		(work-hours in tenths)
(9) Control-time = 1+2+3+7+8		(work-hours in tenths)
(10) Defect-removal-time = 11+6+7+8		(work-hours in tenths)
Process Meeting time		(work-hours in tenths)

Figure 8.12 Blank Data Summary.



Process Meeting Log

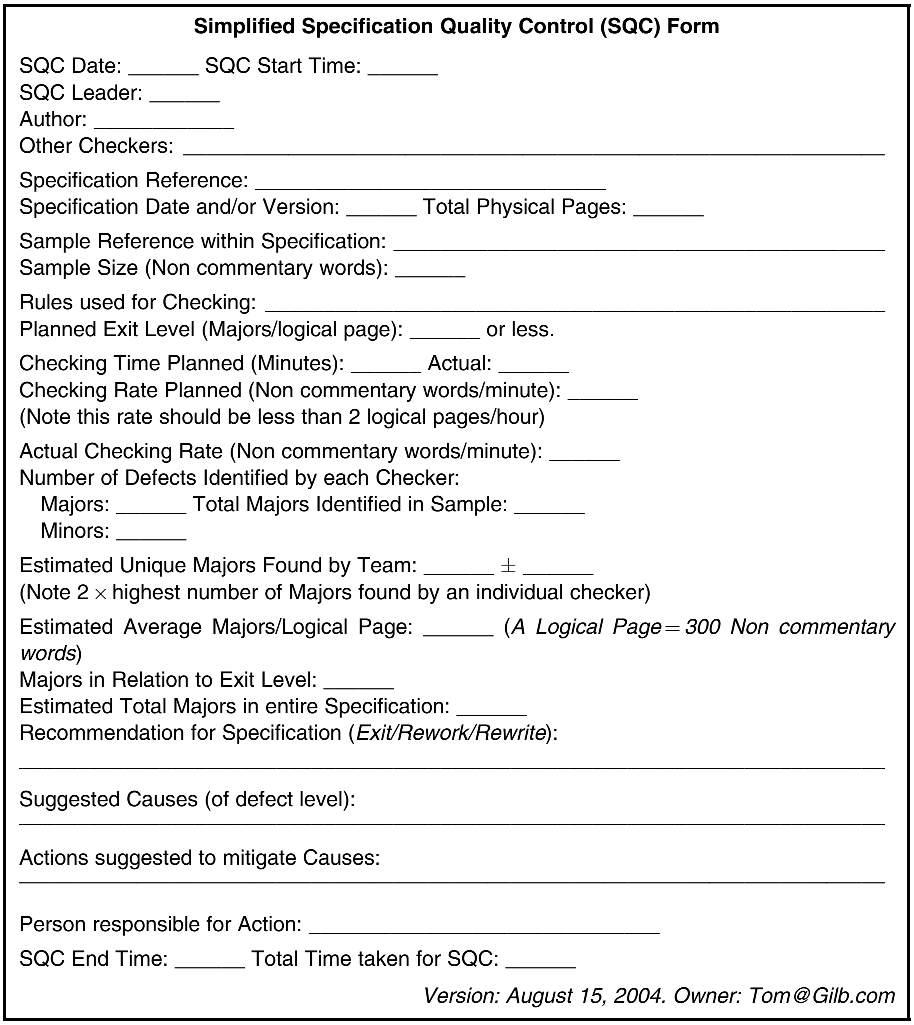
Team Leader _________ Date _________ SQC ID __ Page _ of _

Item	Issue Reference	Cause Class (tick 1)	Root Cause Ideas	Improvement Ideas
1		Communication Oversight Transcription Education		
2		Communication Oversight Transcription Education		
3		Communication Oversight Transcription Education		
4		Communication Oversight Transcription Education		
5		Communication Oversight Transcription Education		
6		Communication Oversight Transcription Education		
7		Communication Oversight Transcription Education		
8		Communication Oversight Transcription Education		
9		Communication Oversight Transcription Education		
10		Communication Oversight Transcription Education		

© Gilb

Start Time	Stop Time	Duration	No. People	Total Cost

Figure 8.13
Blank Process Meeting Log.



Simplified Specification Quality Control (SQC) Form

SQC Date: ______ SQC Start Time: ______
SQC Leader: ______
Author: ____________
Other Checkers: ____

Specification Reference: ______________________________
Specification Date and/or Version: ______ Total Physical Pages: ______

Sample Reference within Specification: ____
Sample Size (Non commentary words): ______

Rules used for Checking: ____
Planned Exit Level (Majors/logical page): ______ or less.

Checking Time Planned (Minutes): ______ Actual: ______
Checking Rate Planned (Non commentary words/minute): ______
(Note this rate should be less than 2 logical pages/hour)

Actual Checking Rate (Non commentary words/minute): ______
Number of Defects Identified by each Checker:
Majors: ______ Total Majors Identified in Sample: ______
Minors: ______

Estimated Unique Majors Found by Team: ______ $\pm$ ______
(Note 2 $\times$ highest number of Majors found by an individual checker)

Estimated Average Majors/Logical Page: ______ (*A Logical Page = 300 Non commentary words*)
Majors in Relation to Exit Level: ______
Estimated Total Majors in entire Specification: ______
Recommendation for Specification (*Exit/Rework/Rewrite*):
____

Suggested Causes (of defect level):
____

Actions suggested to mitigate Causes:
____

Person responsible for Action: ______________________________

SQC End Time: ______ Total Time taken for SQC: ______

Version: August 15, 2004. Owner: Tom@Gilb.com

Figure 8.14
Simplified Specification Quality Control (SQC) form.

8.10 Summary: Specification Quality Control

The basic ideas of SQC are simple:

- "A stitch in time saves nine": fix defects at *early* design stages (DDP), before they cause damage and/or require a costly 'defect removal' process, during test or operation,
- "An ounce of prevention is worth a pound of cure": learn from defects, which have common underlying causes, and continuously improve your work processes (DPP).

Finding Defects

The Defect Detection Process (DDP) is more powerful than similar processes, such as 'checking'[9] (of engineering drawings, as proven at Douglas Aircraft 1988, Boeing 1989 and Thorn EMI 1990 on), 'reviews ', 'walkthroughs ', meetings and management approval. This is mostly due to a series of tactics, the most critical of which are probably the use of a proven *optimum defect-searching time* (optimum checking rate) and, the total focus on *finding and fixing 'major' defects* (which saves time downstream).

Understanding Document Quality

One of the most important opportunities using SQC is to be able to measure the degree to which systems engineering and management documents of all types really do correspond to the required standards of quality. The concepts of 'entry' to, and 'exit' from, all systems engineering and management processes are enabled by our ability to measure 'probable remaining major defects/page' and to decide by estimation if a specification is economic enough to release downstream ('exit'), or economic enough to start work on (allow 'entry'). If necessary, sampling of large documents is an economic way to measure quality levels before making decisions of consequence concerning those documents.

The fact that the SQC process is universally applicable to any readable specification (in any intellectual, administrative, project management, planning, systems engineering, software or user specification task), means that any group of people can use it wherever they want control over quality-in-relation-to-standards. However, SQC has some limitations in understanding 'how well' specifications will work in practice. Even if specifications exit according to any rules you use to analyze them, there can still be catastrophic defects in them in practice. So, we need to use additional methods to see 'how good' a specification is and, if necessary, adjust the specification. *That* is the mission of other tools in this book (like Impact Estimation and Evolutionary Project Management).

The SQC ability to measure quality, in relation to standards, is also important when the standards are a major part of continuous process

[9] Do not confuse with the SQC 'Checking' sub-process! The aircraft factory traditionally used the term 'checking' for a process done by a group of people who specialized in this, called 'checkers.' The process checked engineering drawings against the official engineering drawing specification rules, which were in a large handbook – so large that copies were not given to inform individual engineers what the rules were! In 1988 we proved, with hard data on a large scale, for the engineering directors, that the SQC process was far more effective at finding interesting engineering defects than the traditional 'checking' process. We ended up within the first year with sixty times better quality in terms of rejected and reworked drawings (0.5% versus earlier about 30% reworked).

Possible Purposes For Using SQC

-Reducing Time-to-Delivery
-Measuring the Intelligibility of a Document
-Measuring the effectiveness of engineering specifications
-Measuring the ability of the Process producing the Document to follow best practice rules.
-Enabling Estimation of the Number of Remaining Major Defects
-Identifying Major Defects
-Removing Major Defects
-Preventing consequential 'Downstream' Defects being generated by removing existing Defects
-Improving the Engineering Specification Process (better standards, like rules)
-Improving the SQC Process (better rates, better entry exit conditions, better procedures)
-On-the-Job Training for the Checkers
-Training the Team Leader
-Certifying the Team Leader
-Peer Motivation (getting people to learn, and follow the rules)
-Motivating the Managers (to deal with problems early)
-Helping the Writer (learn to write clearly and have effective ideas)
-Reinforcing Conformance to Standards
-Capturing and Re-using Expert Knowledge (by use of Rules and Checklists)
-Reducing Costs
-Team Building
-Fun – a Social Occasion

Figure 8.15
Possible purposes for using SQC. Any one or several can apply at anytime.

improvement. We can use SQC to measure process improvement efforts! The measurement of defects is a measure of whether people have actually learned, practiced, and understood the continuous improvements intended to increase productivity.

Continuous Process Improvement

The Defect Prevention Process (DPP) exploits grass-root everyday experience with Defect Detection Process (DDP), as well as making use of your data about defects from 'test' and 'field' situations. DPP is the 'engineering and management' version of what Deming (1993) and Juran (1974) taught manufacturing industry, starting in Japan. Experience (Dion 1993; Haley et al. 1995; Kaplan, Clark and Tang 1994) shows that 40% annual productivity improvements are possible, when this is done properly (which is rare).

Chapter
9

Impact Estimation

How to understand strategies

Glossary Concepts

Baseline
Impact Estimate
Scale Impact
Scale Uncertainty
Incremental Scale Impact
Percentage Impact
Percentage Uncertainty
Performance to Cost Ratio
Credibility
Safety Factor
Safety Margin
Safety Deviation
Side Effect
Uncertainty

9.1 Introduction to Impact Estimation

Systems engineers and managers need a reliable way of analyzing how effective their design ideas or strategies are in meeting the requirements. Surprisingly, there are few methods being taught or used to do this. Impact Estimation (IE) is one of these methods. It is the only one that attempts to use any quantified rigor.

The intention of IE is that it helps answer the question of how our design ideas impact all a system's critical performance attributes (such as usability and reliability) and all its resource budgets (such as the financial cost and staff headcount) for implementation and operational running. This question is fundamental to systems engineering.

IE can be used for a wide variety of project purposes. Its most important uses include:

- Comparing alternative design ideas: "What's best?"
- Estimating the state of the overall design architecture: "Have we designed enough?"
- Analyzing risk: "Where are our biggest problems now?"
- Planning and controlling evolutionary project delivery steps: "Is the project on track?"

IE can be used at any organizational level and by different specialist staff roles (such as systems analyst, architect, risk analyst, project manager and purchasing manager) to evaluate any technical or organizational idea. In fact, IE is useful in permitting *integrated assessment* of technical and organizational design ideas. It is specifically helpful in *improving communication* about system design decisions across organizational levels and boundaries.

Impact Estimation Policy

1. All design ideas or strategies which can have a significant impact (5% or more) on any critical performance or cost requirement of a project must be evaluated in an IE table.
2. The design ideas must be specified in sufficient detail and clarity to support IE, irrespective of who would make or evaluate the estimates.
3. An IE table, together with all its related design and requirement specifications, must be quality controlled with respect to all the relevant rules. The level of estimated remaining major defects/page must be low enough to exit and it must be stated (ideally on the cover page of the document).
4. Significant proposed changes to the design ideas or architecture must be accompanied by a quality controlled IE table showing the net impact of the changes.

Figure 9.1
Impact Estimation Policy. Several of my clients have adopted a policy mandating use of IE. This ensures people use the method and helps management (assuming they are IE literate) make more informed decisions about proposed strategies.

IE can be used for a wide variety of purposes including:

1. Evaluating a single design idea. How good is the idea for us?
2. Comparing two or more design ideas to find a winner, or set of winners. Hint: Use IE, if you want to set up an argument against a prevailing popular, but weak design idea!
3. Gaining an architectural overview of the impact of all the design ideas on all the objectives and budgets. Are there any negative side effects? What is the cumulative effect?
4. Obtaining systems engineering views of specific components, or specific performance aspects. For example: Are we going to achieve the reliability levels?
5. Analyzing risk: evaluating a design with regard to 'worst case' uncertainty and minimum credibility.
6. Planning evolutionary project delivery steps with regard to performance, value, benefits and cost.
7. Monitoring, for project management accounting purposes, the progress of individual evolutionary project delivery steps and, the progress to date compared against the requirement specification or management objectives.
8. Predicting future costs, project timescales and performance levels.
9. Understanding organizational responsibility in terms of performance and budgets by organizational function.*
10. Achieving rigorous quality control of a design specification prior to management reviews and approval.
11. Presenting ideas to committees, management boards, senior managers, review boards and customers for approval.
12. Identifying which parts of the design are the weakest (risk analysis). Hint: If there are no obvious alternative design ideas, any 'weak links' should be tried out earliest, in case they do not work well (risk management). This impacts scheduling.
13. Enabling configuration management of design, design changes, and change consequences.
14. Permitting delegation of decision-making to teams. People can achieve better internal progress control using IE, than they can from repeatedly making progress reports to others, and acting on others' feedback.
15. Presenting overviews of very large, complex projects and systems by using hierarchical IE tables. Aim for a one page top-level IE view for senior management.
16. Enabling cross-organizational co-operation by presenting overviews of how the design ideas of different projects contribute towards corporate objectives. Any common and conflicting design ideas can be identified. Hint: This is important from a customer viewpoint; different projects might well be delivering to the same customer interface.
17. Controlling the design process. You can see what you need, and see if your idea has it by using an IE table. For example, which design idea contributes best to achieving usability? Which one costs too much?
18. Strengthening design. You can see where your design ideas are failing to impact sufficiently on the objectives; and this can provoke thought to discover new design ideas or modify existing ones.
19. Helping informal reasoning and discussion of ideas by providing a framework model in our minds of how the design is connected to the requirements.
20. Strengthening the specified requirements. Sometimes, you can identify a design idea, which has a great deal of popular support, but doesn't appear to impact your requirements. You should investigate the likely impacts of the design idea with a view to identifying additional stakeholder requirements. This may provide the underlying reason for the popular support. You might also identify additional types of stakeholders.

*Note: * In 1992, Steve Poppe pioneered this use at executive level while at British Telecom, North America.*

Figure 9.2
Purposes for the use of Impact Estimation. IE can have a wide variety of uses for a systems engineer, planner or manager: it can help from the earliest stages of evaluating potential ideas, strategies, architectures and purchases, to formally presenting proposals to management, to assessing the results of project delivery.

Strategy Comparison: Apples and Oranges

Alternative Strategies

Objectives	Apples	Oranges	
Eater Acceptance From 50% to 80% of People	70%	85%	
Pesticide Measurement Reduce from 5% to 1%	50%	100%	
Shelf-Life Increase from 1 week to 1 month	70%	200%	
Vitamin C Increase from 50 mg to 100 mg per day	50%	80%	
Carbohydrates Increase from 100 mg to 200 mg per day	20%	5%	
Sum of Performance	260%	470%	
Resources			
Relative Cost Local currency	0.50	3.00	
Sum of Costs	0.50	3.00	
Performance to Cost Ratio	5.2	1.57	

"Evidence" for these numbers should, of course, be available on a separate sheet (but not shown here)

Figure 9.3
Comparison of Apples and Oranges using an IE table. IE allows you to compare all kinds of strategies (solutions) against your requirements.

9.2 A Simple Practical Example of Impact Estimation

Now let's consider a practical example and show how you can use the IE approach. Assume you have an objective as follows:

Learning:
Gist: Make it substantially easier for our users to learn tasks <- Marketing.
Scale: Average time for a defined [User Type: Default UK Telesales Trainee] to learn a defined [User Task: Default Response] using <our product's instructional aids>.
Response: Task: Give correct answer to simple request.
Past [Last Year]: 60 minutes.
GN: Goal [By Start of Next Year]: 20 minutes.
GA: Goal [By Start of Year After Next]: 10 minutes.

Imagine you have an initial design idea to satisfy the goals GN and GA:

Handbook: Gist: Write a user handbook to define how to do the tasks.

Now, we could just write the handbook, and hope we shall meet our objectives. But the purpose of IE is to get us to think before we implement. So, let us make an estimate of how effective this idea is. How many minutes will be needed to learn the defined task 'Response' using the handbook? The likely initial answer is, "we cannot possibly know." "Why?" Well, maybe we don't even know the written handbook can, or will, be used by the user. Maybe we don't know if the handbook (assuming it can and will be used) is capable of reducing the learning time compared to last year's training methods (Past level). We might also not have a sufficiently clear and unambiguous definition of the task, Response. The conclusion to this line of thinking is that we need to have a much better design and more detailed specifications in order to make any assertions whatsoever. It is precisely this problem of inadequate design and lack of information that we want to identify and attack by using IE.

Well, let us for this example try a symbolic improvement of the design ideas to meet the goal. We need to identify some alternative design ideas and assess their impact on our Learning goals. We can draw on any previous experience with the use of a design idea. Say, on a different project, the design idea On-line Help had achieved Past [<similar task>] 10 minutes. What do we think based on that? Let us say, we guess a learning time of 10 minutes average (minimum 5 minutes, maximum 15 minutes):

Impact Estimate for impact of On-line Help on Learning = 10 ± 5 minutes? <- Based on <similar design> used by Project A.

We can then express this guess as a 'percentage of the way to the goal.' We must decide on which of the goals, GA or GN? Say, the GA goal of 10 minutes. Well, the guess is also 10 minutes, so we have a design, which appears to get us 100% of the way to our GA goal. The uncertainty, ±5 minutes, is 10% (from Past = 60 minutes to Goal = 10 minutes is 50 minutes improvement). So we can express the impact as either 10 ± 5 minutes (a Scale Impact estimate) or 100% ± 10% (a Percentage Impact estimate).

In practice, we would have to evaluate the effect of all design ideas on all goals and budgets. See Figure 9.5. We are not 'done' until we have satisfied all performance goals (100% or more) within all budgets (100% or less). In the worst case, if the design ideas completely fail to meet the requirements and there are no additional design ideas that could be considered, we have to modify the goals and/or budgets (make 'tradeoffs'). There must be a correspondence between your plans and the realities of what you can actually achieve. Of course, do not lose sight of the fact that the real test is trying out the chosen design ideas in practice to see how they really work in reality.

Design Ideas

On-line Support: Gist: Provide an optional alternative user interface, with the users' task information for defined task(s) embedded into it.
On-line Help: Gist: Integrate the users' task information for defined task(s) into the user interface as a 'Help' facility.
Picture Handbook: Gist: Produce a radically changed handbook that uses pictures and concrete examples to *instruct*, without the need for *any* other text.
Access Index: Gist: Make detailed *keyword indexes*, using *experience* from *at least ten* real users learning to carry out the defined task(s). What do *they* want to look things up under?

Figure 9.4
Brief description of some design ideas to improve learning time.

Table 9.1 An Impact Estimation table showing the impacts of the design ideas described in Figure 9.4 on the Learning objective.

	On-line Support	*On-line Help*	*Picture Handbook*	*On-line Help + Access Index*
Learning 60 minutes <-> 10 minutes				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3 min.	±5 min.	±10 min.	±5 min.
Percentage Impact	110%	100%	60%	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%	±20%?	±10%
Evidence	Project Ajax: 7 minutes	Other Systems	Guess	Other Systems + Guess
Source	Ajax Report, p.6	World Report, p.17	John B	World Report, p.17 + John B
Credibility	0.7	0.8	0.2	0.6
Development Cost	120 K	25 K	10 K	26 K
Performance to Cost Ratio	110/120 = 0.92	100/25 = 4.0	60/10 = 6.0	104/26 = 4.0
Credibility-adjusted Performance to Cost Ratio (to 1 decimal place)	0.92*0.7 = 0.6	4.0*0.8 = 3.2	6.0*0.2 = 1.2	4.0*0.6 = 2.4
Notes: Time Period is two years.	Longer timescale to develop			

Notes: Here it is a case of comparing design ideas. It is not appropriate to assume that the effects of the different design ideas are cumulative. The design idea of Picture Handbook is seen as very cost-effective, but it doesn't on its own meet the goals. Maybe there is a complementary design idea that could be found? On-line Support is seen as achieving the goals (though the safety margin is not extremely comfortable) but, it is not very cost-effective compared to On-line Help and the development timescales need considering. Overall, there is a need to review the long term strategy. Short term, On-line Help seems an ideal design idea to start considering further.

9.3 Language Core: Impact Estimation

The *inputs* to IE include:

- Specified quantified **Performance Requirements** (objectives) **and Resource Requirements**. (This is usually for a specific system/ project deadline, and usually consists of the goals with supporting baseline information, and the budgets.)
- Specified **Design Ideas** with experience data (**Evidence**, **Sources** and basis for **Credibility** assessments).
- Standard **Credibility Ratings** and **Safety Margins**. (These will either exist in rules or policies or they must be decided locally by the project.)

The *outputs* from IE include:

- **IE tables**: 2- and/or 3-dimensional graphical diagram(s).
- **Estimations and calculations for the impacts of each of the specific design ideas on each of the specific goals and each of the specific budgets**:

  - **Scale Impact** and **Scale Uncertainty** values: What estimated impact does a specific design idea have on a specific goal or budget and, what is the margin for error or doubt? A Scale Impact is expressed as a numeric value on the defined Scale (For example, if the scale of measure was in hours, the value could be 10 hours). A Scale Uncertainty is the plus/minus error margin or experience range estimated for the Scale Impact value (for example, ±2 hours). Estimates must be based on experience data; **Evidence**, **Source** and **Credibility** must therefore be stated, or referenced, to support each estimate.
  - **Percentage Impact** and **Percentage Uncertainty** values: What percentage of the required change in a specific goal or budget does a specific design idea provide? For a goal (a performance objective), a Percentage Impact is calculated as the percentage change (that is, the ability to move) from the chosen baseline level (0%) towards a specified target level (100%). (0% would mean there was no change/improvement on the existing past level and 100% would mean the target goal was met exactly. All other percentage estimates are in relation to these two values.) For a budget, a Percentage Impact is the percentage of the budget that is estimated will be consumed or utilized. Percentage Uncertainty values for budgets are calculated in a similar way to goals. *Note: Sometimes it is appropriate to declare an overall Percentage Uncertainty (for example, ±50%) for the whole IE table or specified parts of it.*

  Calculated values for each individual design idea (the 'vertical sums'):

  - **Sum of Performance**: How 'good' is a design idea? Sum of Performance is the sum of all the estimated Percentage Impacts achieved by the design idea across all the performance requirements (objectives). There is also a need to sum the relevant Percentage Uncertainty impacts.

- **Sum of Costs or Sum of Scale Costs**: How costly is a design idea? Sum of Costs and Sum of Scale Costs are the sums of the Percentage Impacts or the Scale Impacts respectively that have been estimated for a specific design idea across all the appropriate budgets. (For example, it is likely to be 'appropriate' to use only the total financial cost figures, though the IE table might also show detailed person work-hours as a 'cost' row.) There is also a need for the sums of the relevant uncertainty impacts (the Percentage Uncertainty and/or Scale Uncertainty values as appropriate).
- **Performance to Cost Ratio**: How cost-effective is a design idea? The performance to cost ratios can be calculated either as Sum of Performance/Sum of Costs or, Sum of Performance/Sum of Scale Costs.

Calculated sums for each individual requirement (the 'horizontal' sums):

- **Sum for Requirement**: Is this requirement likely to be met and what is the margin for error or doubt? Sum for Requirement is the sum of the Percentage Impacts of the selected sets of design ideas on a specific requirement. The sums for the relevant Percentage Uncertainty impacts also need to be calculated.
- **Safety Deviation**: How much risk can be tolerated? This is the deviation of Sum for Requirement from the relevant Safety Margin. A minimum Safety Margin of factor 2 must be assumed by default (this translates to 200% for performance requirements and, to 50% for resource requirements). Include appropriate uncertainty (±) data.

Other IE Process Outputs:

- **Credibility-Adjusted Values**: The sums obtained by repeating all the calculations using the credibility-adjusted estimates (that is after multiplying each estimate with its relevant Credibility).
- **Credibility Averages**: The set of credibility-adjusted values for Sum for Requirement can be averaged to give a figure for the overall likelihood of meeting the requirements. Also the credibility-adjusted Performance to Cost Ratios for the selected design ideas can be averaged. There might be a specified design standard for performance to cost ratios, or stakeholder benefit to cost ratios that has to be exceeded before any budget will be allocated (for example, a specific ratio of Return on Investment (ROI)).
- **Revised Requirements**: Working through an IE table might lead to a revision of expectations, or some new requirements (especially objectives) might well be identified.
- **Revised Design Ideas**
- **Notes and Comments**: It is important to capture the ideas and assumptions that are identified while working through an IE table.
- **Conclusions and Presentations**: The results of analyzing an IE table including risk analysis, gap analysis and recommendations.

Table 9.2 Simple IE table illustrating some of the components of an IE table

Design Ideas-> *Requirements: Goals and Budgets*	*Idea 1 Impact Estimates*	*Idea 2 Impact Estimates*	*Sum for Requirement (Sum of Percentage Impacts)*[3]	*Sum of Percentage Uncertainty Values*[4]	*Safety Deviation*[5]
Reliability 300 <-> 3000 hours MTBF	1950 hr (1650 hr) ±0[1]	1140 hr (840 hr) ±240			
	61% ± 0[2]	31% ± 9%	92%	± 9%	−108%
Usability 20 <-> 10 minutes	19min. (1min.) ±4	14min. (6 min.) ±9			
	10% ± 40%	60% ± 90%	70%	±130%	−130%
Maintenance 1.1M <-> 100 K USdollars/year	1.1M $/Y (0 K$/Y) ±180 K	100 K S/Y (1 M$/Y) ±720 K			
	0% ± 18%	100% ± 72%	100%	±90%	−50%
Sum of Performance[6]	71%	191%			
Capital 0 <-> 1M USdollars	500 K (500 K) ±200 K	100 K (100 K) ±200 K			
	50% ± 20	10% ± 20	60%	±40%	−10%
Sum of Costs[7]	50%	10%			
Performance to Cost Ratio[8]	1.42 (71/50)	19.10 (191/10)			

Notes:

1. Time Period: Within next 12 months.

2. Same Safety Margin of factor 2 has been declared for performance requirements and resource requirements. Factor 2 means minimum planned performance requirements > 200% of target (goal), and maximum planned costs <50% of target (budget).

3. Evidence, Source and Credibility not stated.

Key:

[1] *Scale Impact estimate, (Incremental Scale Impact) and Scale Uncertainty estimate.*

[2] *Percentage Impact estimate with Percentage Uncertainty estimate.*
61% = (1650/(3000 − 300 = 2700)) × 100
31% = (840/(2700)) × 100, ±9% = (240/2700) × 100
10% = (1/(20 − 10)) × 100, ±40% = (4/(20 − 10)) × 100
60% = (6/(20 − 10)) × 100, ±90% = (9/(20 − 10)) × 100
0% = (0/(1.1 M − 100 K)) × 100, ±18% = (180 K/(1,1 M − 100 K)) × 100
100% = (100 K/(1.1 M − 100 K)) × 100, ±72% = (720 K/(1.1 M − 100 K)) × 100

[3] *Sum of Percentage Impacts on a single requirement (Sum for Requirement).*

[4] *Sum of plus/minus Percentage Uncertainty impacts on a single requirement.*

[5] *Statements of deviation from required Safety Margins (Safety Deviation). Value calculated by (Sum for Requirement – Safety Margin). −108% = 92 – 200 (expressed as a negative value)*

[6] *Sum of all performance Percentage Impacts for a single design idea (Sum of Performance).*

[7] *Sum of cost Percentage Impacts for a single design idea (Sum of Costs).*

[8] *Calculation of the ratio of the sum of the percentage performance improvements to the sum of the percentage costs for each design idea (Performance to Cost Ratio).*
The results identify that Idea 2 is better than Idea 1.

9.4 Rules/Forms/Standards: Impact Estimation

Tag: Rules.IE.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Base: The generic rules, Rules.GS and the requirement specification rules, Rules.RS apply.

R1: **Table Format**: The requirements must be specified in the left-hand column. The design ideas must be specified along the top row.

R2: **Requirement**: Each performance requirement (objective) and each resource requirement must be identified by its tag and by a simplified version of the chosen Baseline<->Target Pair (B<->T pair). The B<->T pair should be written under the tag.

Each B<->T pair must consist of two reference points, the chosen baseline (Past) and the planned target (Goal or Budget). Each reference point must be stated as a numeric value or as a tag to a numeric value. The numeric values must be expressed using the chosen Scale for the requirement.

The baseline is stated first as it represents the 0% incremental impact point. Then usually an arrow '<->'. Then the planned target, which represents the 100% incremental impact point.

It must be possible to distinguish between multiple-level specifications for the same Goal or Budget statement. Where necessary, to be unambiguous, use a qualifier or tag the specific baseline and/or target for use in the IE table.

EXAMPLE

Reliability:
Type: Performance Requirement.
Baseline<->Target Pair:
Benchmark Reliability <-> 30,000 hours [USA, Next Year].
Note: Reliability and Benchmark Reliability are tags.

R3: **Qualifiers**: If there is one common set of qualifier [time, place and event] conditions for reaching all targets, this should be explicitly stated in the notes accompanying the IE table. If the qualifiers vary then they must be explicitly stated next to the relevant B<->T pair.

By default, the entire system is implied and no specific conditions are assumed. The deadline time period must always be explicitly stated.

R4: **Design Idea**: Each single column must identify a design idea or set of design ideas that could be implemented as a distinct Evo step. Each design idea must be identified by its tag. Multiple tags may be specified as a set of design ideas in a single column. All tags must be supported by a design specification, which must exist in the supporting documentation and must be sufficiently detailed to allow impact estimations to the required level of accuracy. As a minimum, each design specification must be sufficiently detailed to permit financial cost to be estimated to within an 'order of magnitude.'

R5: **Scale Impact**: For each goal or budget, the Scale Impact is the estimated or actual performance or cost level respectively (expressed using the relevant Scale) that is brought about by implementing the design idea(s) in each column.

R6: **Percentage Impact**: The Percentage Impact is a percentage (%) value derived from the Scale Impact (see Rules.IE.R2). An estimate of zero percent, '0%,' means the impact of the implementation of this design idea is estimated to be equal to the specified baseline level of the objective. '100%' means the specified target level would probably be met exactly and on time. All other percentage estimates are in relation to these two points. Note: In an IE table, it is acceptable to specify either Percentage Impacts and/or the Scale Impacts (the absolute values on the defined scale of measure). *Examples: 60%, 4 minutes.*

R7: **Uncertainty**: The ± Uncertainty (based on the evidence experience borders) of the Scale Impact estimate shall normally be specified. Percentage Uncertainty values are then calculated in a similar way to the Percentage Impacts. *Example: 60% ± 20%.* Usually, the uncertainty values are calculated individually for each cell. An exception to this occurs when some overall uncertainty (such as ±50%) is declared for the whole table or specified parts of it. Another more fundamental exception can be when a decision is made to defer dealing with uncertainty data.

R8: **Evidence**: Each estimate must be supported by facts that credibly show how it was derived. Numbers, dates and places are expected. If there is no evidence, a clear honest risk-identifying statement expressing the problem is expected (such as 'Random Guess' or 'No Evidence'). The exact source of the evidence must also be explicitly stated. Note: Reference to a specific section of a document is permitted as evidence.

R9: **Credibility**: The evidence, together with its source, must be rated for its level of credibility on a scale of 0.0 (no credibility) to 1.0 (perfect credibility).

The relevant standard Credibility Ratings Table must be considered for use. Explanation must be given if alternative ratings are chosen.

R10: **Completeness**: All IE cells (intersections of a design idea and a requirement) must have a non-blank statement of estimated impact. This must be given as a numeric value using the relevant Scale units, or as a Percentage Impact as assessed against the defined Baseline <->Target Pair, or both. If there is no estimate, then a clear indication of this must be given.

R11: **Calculations**: All the appropriate IE calculations must be carried out and the arithmetic must be correct. Hint: Using an application, such as a spreadsheet, helps! The IE calculated values include:

- Percentage Impact: See Rule R6.
- Percentage Uncertainty: See Rule R7.
- Sum of Performance: For each design idea, an algebraic sum of its Percentage Impacts on all the performance requirements. (A 'vertical' sum.)
- Sum of Costs: For each design idea, an algebraic sum of all its Percentage Impacts on the selected resource requirements. ('Selected' as it might well not make sense to sum all the costs represented in an IE table.) (A 'vertical' sum)
- Sum of Scale Costs: For each design idea, an algebraic sum of all its Scale Impacts on the selected resource requirements. (A 'vertical' sum.)
- Performance to Cost Ratio: The performance to cost ratios are calculated using either (Sum of Performance/Sum of Costs or Sum of Performance/Sum of Scale Costs).
- Sum for Requirement: For each requirement, an algebraic sum of all the Percentage Impacts for the simultaneously applicable and compatible design ideas. (A 'horizontal' sum.)
- Safety Deviation: For each requirement, subtract the Safety Margin from the Sum for Requirement. The relevant standard safety margin must be considered for use. Explanation or justification must be given if an alternative safety margin is chosen for use. By default, a standard safety margin of factor 2 (200% for performance requirements, 50% for budgets) will be used. For example, if the required safety margin is 200% and Sum for Requirement for a performance requirement is 120%, then "–80%" is the deviation to be displayed. (A 'horizontal' sum.)
- Calculate all the relevant (±) uncertainty values. Base this on best case and worst case observations or estimates.

Table 9.3 Example of a Credibility Ratings Table

Credibility Rating	*Meaning*
0.0	Wild guess, no credibility
0.1	We know it has been done somewhere
0.2	We have one measurement somewhere
0.3	There are several measurements in the estimated range
0.4	The several measurements are relevant to our case
0.5	The method used to obtain the several relevant measurements is considered reliable
0.6	We have used the method/design/idea/strategy in-house
0.7	We have reliable measurements for the design idea in-house
0.8	Reliable in-house measurements correlate to independent external measurements
0.9	We have used the idea on this project and measured it (Evo step, pilot and field trial)
1.0	Perfect credibility, we have rock solid, contract-guaranteed, long-term and credible experience with this idea on this project and, the results are unlikely to disappoint us

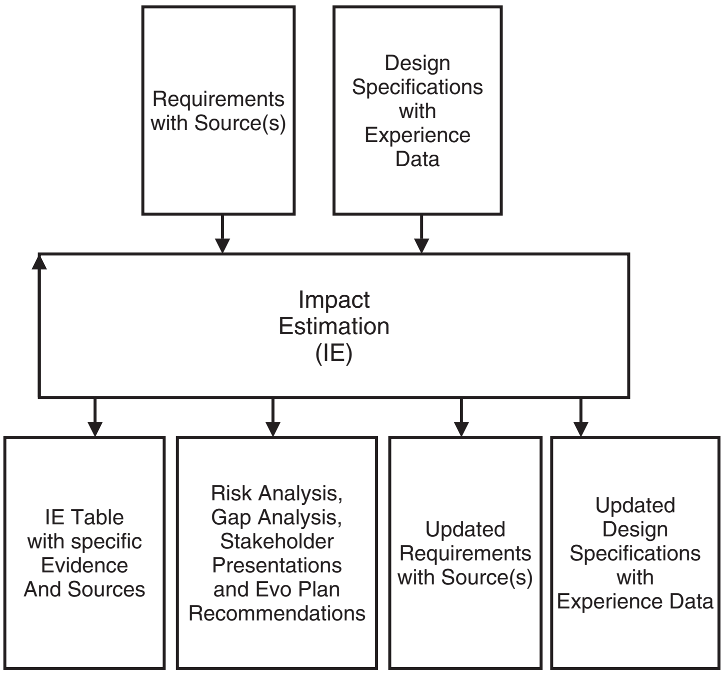

Requirements with Source(s)
Design Specifications with Experience Data
Impact Estimation (IE)
IE Table with specific Evidence And Sources
Risk Analysis, Gap Analysis, Stakeholder Presentations and Evo Plan Recommendations
Updated Requirements with Source(s)
Updated Design Specifications with Experience Data

Figure 9.5
Overview of the Impact Estimation Process.

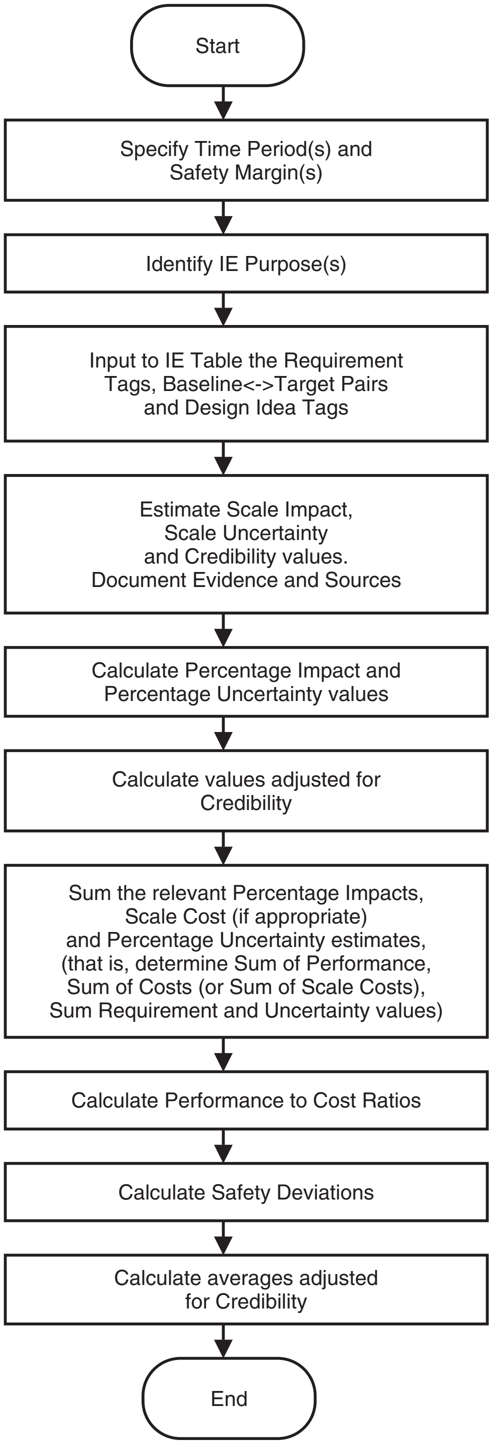

Start
Specify Time Period(s) and Safety Margin(s)
Identify IE Purpose(s)
Input to IE Table the Requirement Tags, Baseline<->Target Pairs and Design Idea Tags
Estimate Scale Impact, Scale Uncertainty and Credibility values. Document Evidence and Sources
Calculate Percentage Impact and Percentage Uncertainty values
Calculate values adjusted for Credibility
Sum the relevant Percentage Impacts, Scale Cost (if appropriate) and Percentage Uncertainty estimates, (that is, determine Sum of Performance, Sum of Costs (or Sum of Scale Costs), Sum Requirement and Uncertainty values)
Calculate Performance to Cost Ratios
Calculate Safety Deviations
Calculate averages adjusted for Credibility
End

Figure 9.6
Creating an IE table.

R12: **Credibility-Adjusted Calculations**: *Do not get carried away with these credibility calculations if they are not adding significant value. They are meant to force you to think about risks.*

Multiply all the values for Scale Impact, Percentage Impact and Uncertainty by their Credibility. Repeat the calculations described in Rule R11 using the credibility-adjusted values.

For each Design Idea: Calculate the credibility-adjusted average (Design Idea Credibility Average) by dividing the sum of its credibility-adjusted Percentage Impacts for all the performance requirements by the number of performance requirements being considered. (A 'vertical' sum.)

For each Requirement: Calculate the credibility-adjusted average (Requirement Credibility Average) by dividing the sum of its credibility-adjusted Percentage Impacts for all the relevant design ideas by the number of relevant design ideas. (A 'horizontal' sum.)

9.5 Process Description/Standards: Impact Estimation

Process: Impact Estimation

Tag: Process.IE.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Entry Conditions

E1: The Generic Entry Conditions apply. The main input documents are the requirement specification and the design specifications.

Note: It is extremely important that the requirement specification is SQC exited. Note also that the Credibility of the Evidence and Source(s) will be independently rated during IE, regardless of whether the design specifications are SQC exited.

Procedure

P1: Identify your 'purpose' for the IE table. Decide how to use the table for your defined purposes. Are you using IE for 'self-analysis,' 'presentation to authorities,' 'control of design engineering or planning process,' 'project control,' 'comparison of alternatives' or others? The purpose and audience determine what you do with the table and how rigorous and formal you are. *See Figure 9.2, 'Purposes for the use of IE'.*

P2: Use the rules, Rules.IE, to fill out an IE table to the best of your ability. This implies that all the IE calculations are done, perhaps using a software application (see footnote 4 at the end of Section 9.9).

P3: Be honest and open. Document where insufficient information is available, and where guesses are being made. Make liberal use of '?' and other 'uncertainty' indicators. Remember, the IE table is there to help you see potential problems, not to cover them up!

P4: Analyze Risks. Specify risks in your report or presentation. For example, as footnotes to the IE table. Try the following:

- Study the requirements again. Which ones are 'shaky'? Look for '<fuzzy>', '?', dubious sources and admitted guesses.
- Study the design specifications again. Are they really specific and detailed enough to merit the estimates? Is the evidence really good enough to 'stand up in a court' of skeptics?
- Study the table itself for gaps to targets. For example, consider the gaps to the goal targets. Also look at the safety deviations. Document any gap problems that you identify and suggest actions.

P5: Identify the areas that deserve more time-demanding analysis, and work more on them. For example, you should select areas of the table with low credibility, high uncertainty and large shortfalls in meeting goals or budgets.

P6: Make improvements and changes to requirements, designs, and evidence. Re-calculate the table.

- The owners of these requirements and design must be involved ultimately. Are you being ambitious enough?

P7: Decide which issues need to be settled 'in the field' by (Evo steps, prototypes, market trials, field experiments).

- Make specific recommendations about which areas need early practical measurement. Show the estimated impacts of implementation of the different design ideas.

P8: Decide on presentation. Topics you should consider covering include the level of IE table, the requirement hierarchy, key 'focus' issues, graphics, alternative design ideas, risk analysis and suggested actions.

- Bring out the main conclusion. Bring out the risks and dangers. Show the effect of any suggested alternatives.

P9: Make presentations to (colleagues, formal reviews, stakeholders, experts, key managers).

Hint: Choose to present first to informal friends, rather than making a fool of yourself by lack of preparation in front of your managers and stakeholders.

Exit Conditions

X1: The Generic Exit Conditions apply. The IE table and all its related data (such as evidence) shall exit from Specification Quality Control (SQC) with no more than one remaining major defect/page.

- Formally, the table can only be used in other processes when it has exited from SQC. In practice, the purpose and the audience, determine what you are going to demand as exit conditions. The precise exit conditions need to be defined locally.

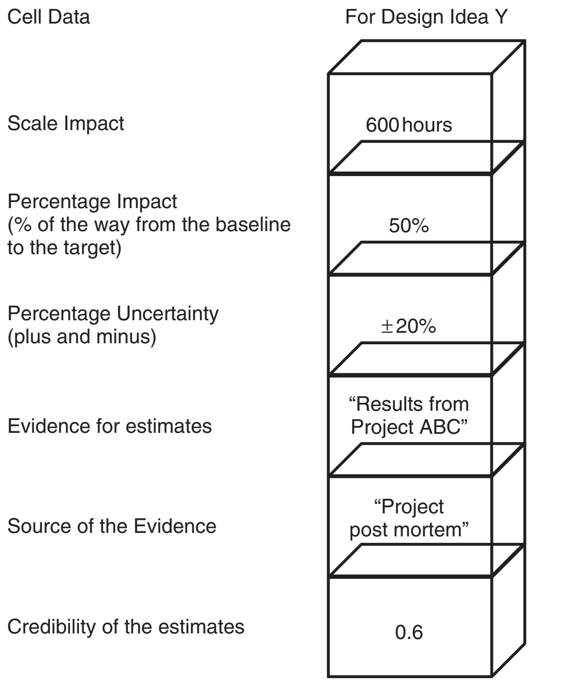
9.6 Principles: Impact Estimation

1. **The Principle of 'Words being difficult to weigh'**
   Non-numeric estimates of impact are difficult to analyze and improve upon. A design idea described as 'excellent' could actually be worse than another merely described as 'good.'
2. **The Principle of 'Doubtful digits are better than none'**
   A bad numeric estimate, and its definition, can still be systematically criticized and improved. In fact, a random number is a better starting estimate than flowery, descriptive words.
3. **The 'Evident' Principle**
   Estimates without sources, evidence and credibility are not evident.
4. **The Principle of 'Uncertainty in no uncertain terms'**
   The uncertainty estimate is at least as important as the main estimate.
5. **The Principle of the 'Seat Belt'**
   A safety margin is as necessary with uncertain estimates, as a seat belt is with uncertain traffic.
6. **The Principle of 'Profitable Proposals'**
   The value of an idea is how well it meets objectives. The net value considers the costs too.
7. **The Principle of 'the Swiss Army Knife'**
   Impact Estimation is a multi-purpose method. It can help you in many situations: to evaluate, to compare, to present, to argue, to destroy, to find weaknesses, to cut fat, to see risk, to prioritize, to sequence and more.

8. **The Principle of 'Always Useful'**
   Impact Estimation can assist a project throughout its lifecycle – from identifying requirements to assessing feedback data from implemented systems.

9. **The Principle of 'Multiplicity'**
   When stakeholders have multiple requirements, then we need to evaluate multiple design options against all those requirements including considerations of value, in order to make a reasonable choice.

10. **The Efficiency Principle**
    When real life has many stakeholder values, and many cost constraints, then evaluation of designs (strategies) must be done with respect to *both* the values and the costs.


Cell Data
For Design Idea Y
Scale Impact
600 hours
Percentage Impact (% of the way from the baseline to the target)
50%
Percentage Uncertainty (plus and minus)
±20%
Evidence for estimates
"Results from Project ABC"
Source of the Evidence
"Project post mortem"
Credibility of the estimates
0.6

Note other options include:

- Percentage Impact towards the Fail levels (Levels of no failures)
- Percentage Impact towards the Survival levels (Levels for survival)
- Other Percentage Impacts towards the Goal/Budget levels (target levels) for other qualifier conditions. (For example, different dates)
- Owner of estimate: Tom

Figure 9.7
The Data in an IE Cell for a Performance Attribute.

9.7 Additional Ideas: Impact Estimation

Understanding Mathematical Inaccuracy

Let me stress that IE provides only rough, practical calculations. Adding impacts of different, independent estimates for different design ideas, which are part of the same overall architecture, is dubious in terms of accuracy. There are bound to be interactions, which we are unable to predict in advance of implementation.

This admission of mathematical inaccuracy often annoys people; on one hand, I'm demanding numeric values and, on the other, I'm admitting to a lack of accuracy! There is no absolute defense for this, apart from saying we can only try our best; quantitative values far better convey understanding than words and they permit calculations to be carried out.

Let me add an additional cautionary note that I expect IE estimates only to be used as a rough indicator to help designers spot potential problems or select design ideas. Any real estimation of the impact of many design ideas needs to be made by real tests; ideally, by measuring the results of early evolutionary steps in the field.

Level of Detail

Understanding your specific purposes for using IE is key to how you actually use the method. These purposes determine how rigorous and formal you are. If you are using IE in brainstorming mode to generate new design ideas and to check you have the right set of requirements, then rough numeric estimates will suffice. If you are establishing which are the most cost-effective design ideas, then more detailed impact estimations will be necessary.

Coping with Interactions amongst Design Ideas

Considering Side Effects

Negative impacts do occur! It is fairly common for a design idea to impact on certain objectives very positively and yet negatively on others.

Dealing with Alternatives

Take care that you are not adding together the percentage impacts of mutually exclusive design ideas.

Dealing with Dependencies

Design ideas can be dependent on each other or their impacts can differ depending on what other design ideas have been implemented in *advance* of them, or even those implemented *later*. Consider grouping dependent design ideas into a set and evaluating as a set within an IE table.

Priority Management within IE

People often ask why I don't use 'ranking' or subjective weights for priority. The answer is that IE handles priority implicitly. You in-build the priorities when you specify the required performance levels over time (Goal levels with time conditions).

Note, this does not mean that you don't discuss priority. You do! You definitely need to understand the priorities when setting and modifying your requirements.

Can We Always Use the Stated Requirements to Determine Priority?

The stakeholder value derived from meeting a goal or other requirement, wholly or partly, has also to be considered. There are many factors, for example, meeting one goal can be very much more rewarding than meeting another, the value derived can vary from stakeholder to stakeholder, or the value can vary according to where the design idea is delivered. So, the question arises, should the project manager decide the priority by looking at the goals they have been given officially, or should they somehow try to figure out what the consequential value is for satisfying a requirement, and get closer to a more realistic priority?

Here are some possible answers:

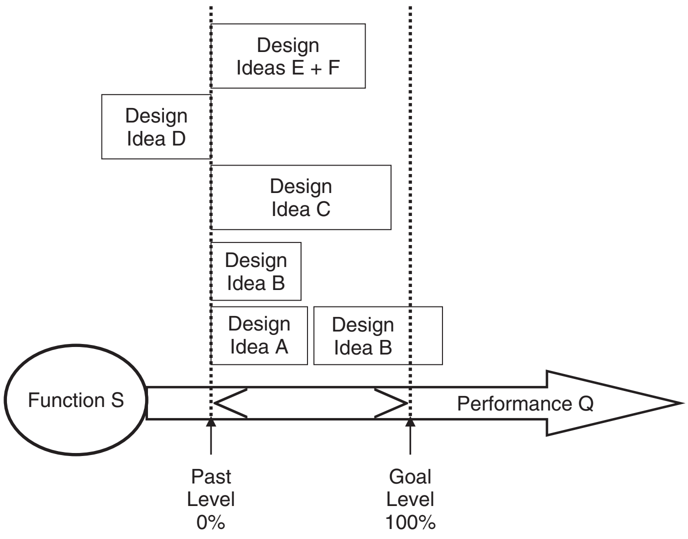
- they should stick to their official goals and other requirements
- they should look to any vision statement(s) and policy statement(s) for direction. If the requirements are not good enough to motivate them in the right direction (that is, do not live up to the vision and policy statement(s)), then this may be an indicator that they should get a reformulated set of requirements at the appropriate level, which reflect value better.
- they can ask key stakeholders exactly which of the unfinished requirements should have priority 'this week' (a common practice in Evolutionary Project Management).

Priority for a designer or manager is to reach the stated goals (performance targets) within the stated budgets (for people, time and money) under the stated conditions. Efforts must be focused towards trying to make the maximum progress, in the direction of immediate goals, at all times. If a specific goal has priority, it has claim on our resources (our budgets) for satisfying that goal.

This is not a simple static problem. Priority changes when any of the following change:

- The design or implementation distance to 'survival levels' (Survival) for a specific performance goal
- The distance to 'success levels' (Goal) for a specific performance goal
- The availability of a given type of resource (for example, if you don't have 'money', then use 'time' instead)
- The uncertainty of an idea: the factors effecting this are evidence, sources, plus/minus uncertainty and the credibility rating.

Such changes occur as a result of the order of implementing design ideas, feedback from the field and changes in the business



All proposed design ideas are supposed to contribute to our ability to reach the planned performance levels within the planned resource levels.

For a specific requirement, Performance Q:

- *Design Idea A is estimated to move us halfway towards a stated goal.*
- *Design Idea B, implemented after Design Idea A, is estimated to bring us the rest of the way, and perhaps give more than our goal.*
- *Design Idea B implemented first, before Design Idea A, is not as effective for this performance attribute.*
- *Design Idea C almost delivers all the required improved performance level on its own.*
- *Design Idea D has a negative effect on his performance attribute.*
- *Design Idea E and Design Idea F are totally dependent on each other and must therefore be considered together.*

  Insight: IE allows us to evaluate partial solutions in various combinations, and pick a satisfactory combination.

  Recommendation: Use IE to look at combinations of solutions, so that your selection is better.

Figure 9.8
Design Idea Contributions.

environment; change in priority is almost inevitable. Priorities vary depending on the 'gaps' between the current level and the target level; the 'larger' a gap in relation to the other gaps, then the more likely it is to demand attention. We can use an IE table as a tool for determining, calculating and visualizing our current priorities.

It is our decision how we manage our priorities. We can use the IE table to manage both the initial design and implementation phases of projects, and ensure that projects are tailored to our current priorities.

Managing Risk: Building in Safety Margins

Priority management (above) is one way of managing risk. However, IE also has an additional mechanism – the use of safety margins. By explicitly designing to overreach your requirements, you can better ensure that you actually reach the requirements. Note that just because you have additional design ideas, does not mean that you have to implement them all!

Highlighting System Failure and System Survival Levels

Another way to control risk is to monitor it more explicitly by using Fail levels or Survival levels, rather than Goal levels, in IE tables. Fail levels reflect the levels of requirements that must be achieved to avoid any project failure. Survival levels reflect the levels of requirements that must *all* be reached for the project to survive. Obviously, when working with the project critical values, the choice of Safety Margin(s) becomes a crucial issue.

9.8 Further Example/Case Study: IE Table for US Army Personnel System Long Term Planning

Here are extracts from a larger study to show you use of the IE method in the 'real world'.

Table 9.4 was produced during a study of the improvement of a US Army Personnel system. The requirements (left column) were specified in detail and quantified. A sample of the Customer Service objective is given below to give the reader some idea of this detail. Notice that as well as the stakeholder objectives being evaluated on this chart, two of the cost aspects for the proposed strategies (design ideas) are also estimated. This makes it possible to see the relative 'bang for buck' of each strategy (by calculating the performance to cost ratio). Comparison

Table 9.4 Example of a real Impact Estimation table from a pro-bono client (US DoD, US Army, Persincom).

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerment*	*Principles of IMA Management*	*Business Process Re-engineering*	*Sum Requirements*
Customer Service ? <->0 Violation of agreement	50%	10%	5%	5%	5%	60%	185%
Availability 90% <-> 99.5% Up time	50%	5%	5–10%	0%	0%	200%	265%
Usability 200 <-> 60 Requests by Users	50%	5–10%	5–10%	50%	0%	10%	130%
Responsiveness 70% <-> ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity	45%	60%	10%	35%	100%	53%	303%
3:1 Return on Investment	50%	5%	75%	45%	15%	61%	251%
Morale 72 <-> 60 per month on Sick Leave							
Data Integrity 88% <-> 97% Data Error %	42%	10%	25%	5%	70%	25%	177%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0%	60%	160%
Requirement Adaptability ? <-> 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M <-> ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS <-> 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
Sum of Performance	*482%*	*280%*	*305%*	*390%*	*315%*	*649%*	
Money % of total budget	15%	4%	3%	4%	6%	4%	36%
Time % total work months/year	15%	15%	20%	10%	20%	18%	98%
Sum of Costs	*30*	*19*	*23*	*14*	*26*	*22*	
Performance to Cost Ratio	*16:1*	*14:7*	*13:3*	*27:9*	*12:1*	*29:5*	

of these performance to cost ratios can be used to decide what to invest in initially (in the early stages of the change process). The strategies were also detailed; only the strategy tag is given at the top of the table. One strategy, 'Technology Investment' is detailed at the Gist level below. The estimates are made in round numbers (nearest 5%). In the full study, Evidence and Sources were given. This was the first time anybody we had contact with there had seen an Impact Estimation table. The General insisted that the analysis and presentation work were taken seriously and done to a reasonable standard.

EXAMPLE

Customer Service: "An example of one of the objectives defined."
Gist: Improve customer perception of quality of service provided.
Scale: Violations of Customer Agreement per Month.
Meter: Log of Violations.
Past [Current Date]: <number of violations> <- Management Review on State of Persincom.
Record [NARDAC]: 0? <- NARDAC Reports [This Year].
Fail: <better than Past> <- CG.
Goal [By End of This Year, Persincom]: 0 "Go for the Record" <- Group SW.

EXAMPLE

Technology Investment: "An example of one of the strategies defined."
Defined As: Exploit investment in high return technology.
Impacts: Productivity, Customer Service.

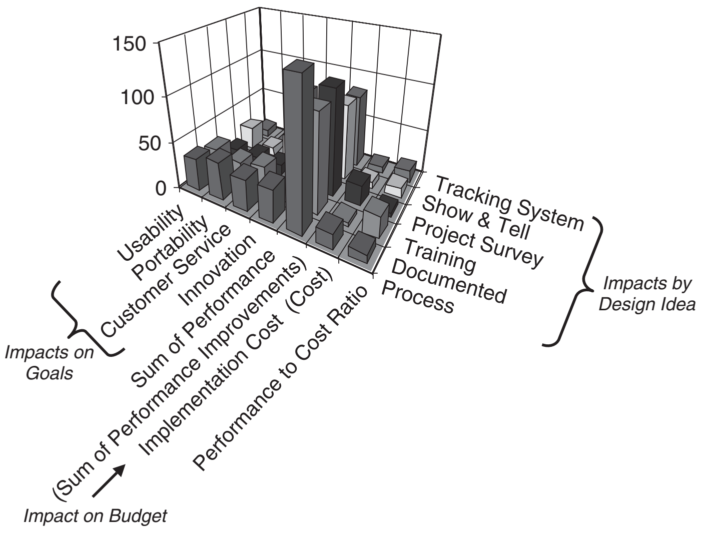
9.9 Diagrams/Icons: Impact Estimation

Presentation of IE Tables

Always consider your audience when presenting IE tables. It is very easy to present too much detail at once. If you are presenting to management, you must use a high-level representation of the IE table. However, always have the detailed version available to support their more searching 'tough' questions!

One possible way to simplify the IE results is to interpret the numeric values into, say, stars with a one to five rating. This works well in a meeting when there is little time.

Another approach is to use the performance to cost ratios and credibility-adjusted averages. Once management understands how these values are calculated, this can be a very rapid way of summarizing the key points.



Question: Look at Figure and try to identify the most cost-effective design idea and how much of the required performance attributes it is likely to deliver (see bottom of page for answer).

Figure 9.9
'Skyscraper' Representation of IE results (a 3-dimensional bar chart by spreadsheet). The figure shows a '3-dimensional' example of an IE table. This gives you an idea of the kind of useful information that an IE table combined with spreadsheet software can provide. Design ideas are along one axis and, performance targets (goals) and cost targets (budgets) are along another. The third axis graphically compares the levels of various types of impact (for example, contributions towards performance goals and performance to cost ratios).

Software Tools Supporting IE

Impact Tables are well suited to spreadsheet software. It is a major benefit to have the calculations automatically worked out and immediately available for analysis. It is also easy to produce pleasing graphics.[1]

Answer: The design idea of providing 'Training' is the most cost-effective. However, on its own it doesn't deliver sufficient levels of performance to be sure of the project's success. Other design ideas should be considered to supplement it, such as 'Tracking System.'

[1] My son, Kai Thomas Gilb has produced a simple working prototype using Microsoft Excel. We often use it for live demonstrations in the classroom. It is free and available at our website, www.Gilb.com. Some clients have made IE tools using Microsoft Access, which has a more pleasing human interface than Excel for entering data. We reckon the reader can easily make their own IE application from available software.

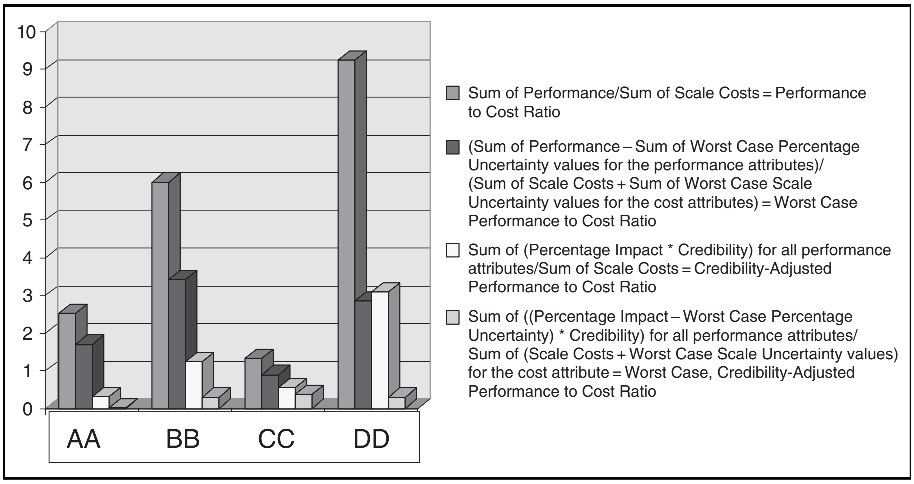
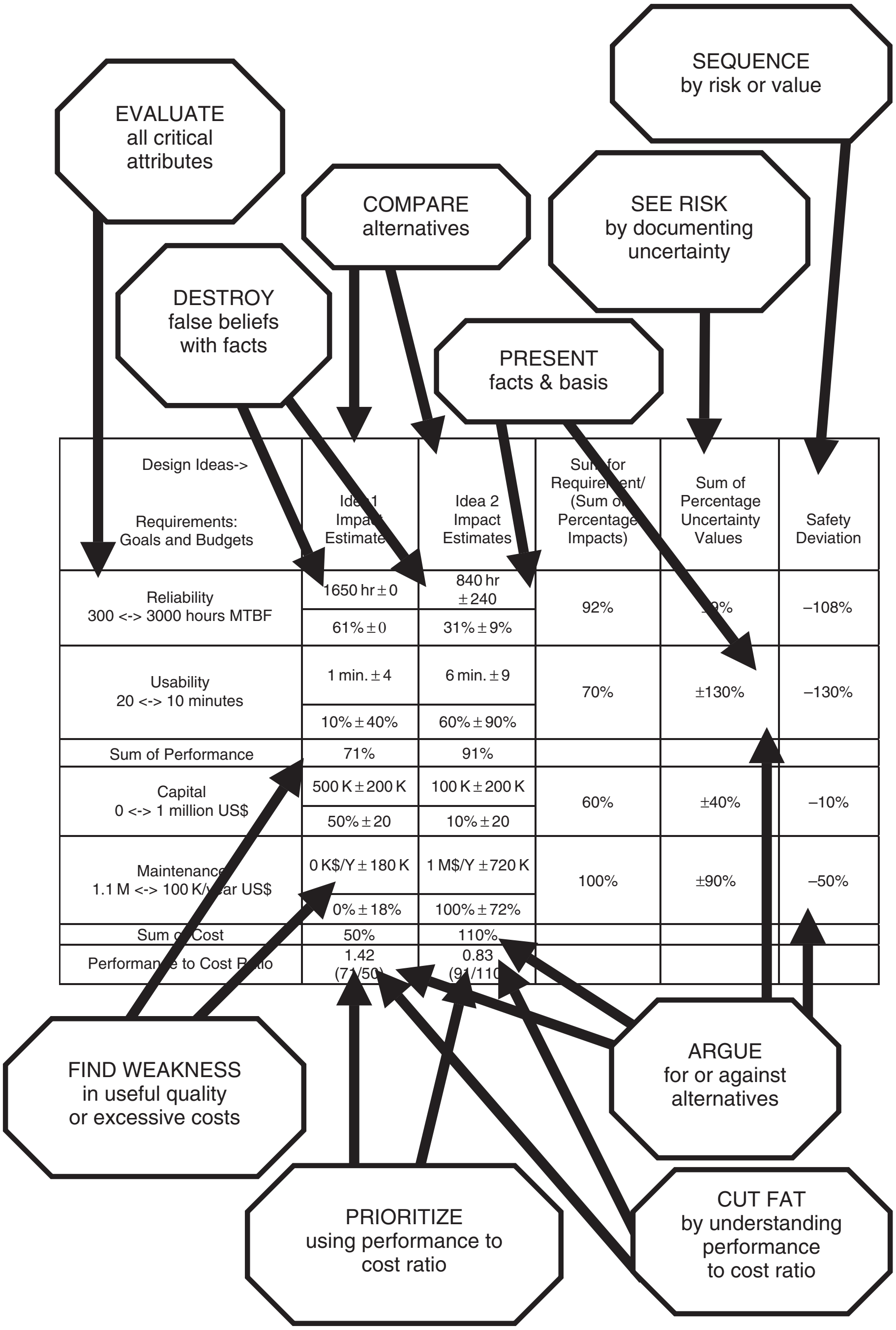


Figure 9.10
Impact Estimation Analysis of worst cases using Uncertainty and Credibility.

Concept	**Keyed Icon**
Impacts	->
Scale Impact	-I-I-.->
Percentage Impact	%.->
Impact Estimate	->.#
Cell	#
Side Effect	*.->
Uncertainty	±?
Percentage Uncertainty	%.±?
Scale Uncertainty	-I-I-.±?
Baseline	0%
Baseline to Target Pair	<->
Credibility	±?.#
Safety Factor	X
Safety Deviation	X.±
Sum of Performance	Σ.O+
Sum of Costs	Σ.-O
Sum for Requirement	Σ.[@]
Performance to Cost Ratio	+%.-%

Figure 9.11
Keyed Icons for Impact Estimation.


SEQUENCE by risk or value
EVALUATE all critical attributes
COMPARE alternatives
SEE RISK by documenting uncertainty
DESTROY false beliefs with facts
PRESENT facts & basis
FIND WEAKNESS in useful quality or excessive costs
ARGUE for or against alternatives
PRIORITIZE using performance to cost ratio
CUT FAT by understanding performance to cost ratio

Design Ideas-> Requirements: Goals and Budgets	Idea 1 Impact Estimates	Idea 2 Impact Estimates	Sum for Requirement/ (Sum of Percentage Impacts)	Sum of Percentage Uncertainty Values	Safety Deviation
Reliability 300 <-> 3000 hours MTBF	1650 hr ± 0	840 hr ± 240	92%	±9%	–108%
	61% ± 0	31% ± 9%			
Usability 20 <-> 10 minutes	1 min. ± 4	6 min. ± 9	70%	±130%	–130%
	10% ± 40%	60% ± 90%			
Sum of Performance	71%	91%			
Capital 0 <-> 1 million US$	500 K ± 200 K	100 K ± 200 K	60%	±40%	–10%
	50% ± 20	10% ± 20			
Maintenance 1.1 M <-> 100 K/year US$	0 K$/Y ± 180 K	1 M$/Y ± 720 K	100%	±90%	–50%
	0% ± 18%	100% ± 72%			
Sum of Cost	50%	110%			
Performance to Cost Ratio	1.42 (71/50)	0.83 (91/110)			

Figure 9.12
Multiple Purposes for IE. Impact Estimation serves many purposes. Here are some headlines and some symbolic pointers to the parts of the IE table which influence these purposes. A list of the main purposes can be found in Figure 9.2.

9.10 Summary

IE is a practical method that can be used throughout the entire lifecycle of a project to help identify and evaluate design ideas against system requirements. Specifically, IE promotes better, more informed design decisions as:

- it forces people to numerically evaluate the impact of design ideas and to provide evidence to support their estimates
- it helps communication about the key elements of the system design; the objectives, the budgets and the design ideas
- it provides a means of understanding and dealing with priority and risk.

In fact, the main problem currently facing people using IE tables is the lack of quantitative data. To make a start, we can use our practical experience data. However, there is a general need to gather more objective data about our technologies. Historically, the emphasis has been solely on cost data.

Chapter
10

Evolutionary Project Management

How to Manage Project Benefits and Costs

Glossary Concepts

Evolutionary Project Management
Gap
Step (or Evo Step)
Result Cycle
Backroom
Frontroom
Before
After
Dependency

10.1 Introduction to Evolutionary Project Management

In 1994, the US Department of Defense issued a temporary military standard, MIL-STD-498, which explicitly supported the use of evolutionary project management (Evo). It also supported the related concept that projects do not initially have the 'final and correct user requirements' specified. This standard has now been evolved into civil standards (such as IEEE standards) and is continuing to influence new standards. Such recognition for Evo is deserved as it has probably the best track record of any known project management method (Larman and Basili 2003).

Practical Experience with Evolutionary Project Management

Surprisingly, many project cultures have little formal knowledge of Evo, even though it has been in use since the 1960s. The first documented large-scale industrial use of Evo was from 1970 to 1980 and on, within IBM Federal Systems Division (IBM FSD, later owned by Loral and Lockheed Martin). Working within the military and space sector, they had complex 'high-tech' projects requiring *state-of-the-art* performance with *fixed* financial budgets and *fixed* deadlines. These extreme requirements drove them into developing methods known as 'Cleanroom', which included using Evo. Harlan Mills reported on these early IBM FSD experiences as follows:

> Ten years ago general management expected the worst from software projects – cost overruns, late deliveries, unreliable and incomplete software. Today, management has learned to expect on-time, within budget deliveries of high-quality software. LAMPS . . . a 4 year . . . 200 person-years (project was delivered) in 45 incremental deliveries. Every one of those deliveries was on time and under budget. [The] NASA space program . . . 7,000 person-years software development . . . few late or overrun [budgets] . . . in . . . [a] . . . decade, and none at all in the past four years.
>
> *Harlan Mills (Mills 1980: reprinted (IBMSJ 1999))*

Harlan Mills told me that it was precisely the 'fixed deadline' and the cost situation of 'lowest bidder wins', which led to the development of Evo. He also told me that their model for Evo was the way in which intelligent military and civil rockets move towards their targets using feedback and control mechanisms.

More recently, Hewlett-Packard has also publicly documented the benefits of Evo (Cotton 1996; May and Zimmer 1996; Bronson 1999; Upadhyayula 2001; MacCormack 2001). Evo has been in use within the organization since at least 1988.[1]

> The evolutionary development methodology has become a significant asset for Hewlett-Packard software developers. Its most salient, consistent benefits have been the ability to get early, accurate, well-formed feedback from users and the ability to respond to that feedback.
>
> *Elaine May and Barbara Zimmer, Hewlett-Packard (May and Zimmer 1996, Page 44).*

Microsoft has also been documented as using Evo extensively (MacCormack 2001; Cusumano and Selby 1995). The Open Source Methods (like Linux) (Maier and Rechtin 2002) and Agile Software Development methods (Cockburn 2002; Abrahamsson et al. 2002) have also clearly demonstrated the power of Evo in delivering good software rapidly.

The use of Evo is also proven within engineering processes. For example in 1988, this author consulted with over 25 projects (about 120 aircraft design engineers) at Douglas Aircraft. Of course, new aircraft did not fly the next week (most of the projects were modifications and upgrades, or integrating new components). But, real results capable of giving useful feedback were delivered to real stakeholders in weekly increments. Management approved each Evo step in advance. They found the method was practical, low risk and they could not resist seeing results fast.

Underlying Principles of Evolutionary Project Management

The underlying principle of Evo is the Plan-Do-Study-Act cycle (PDSA cycle). In other words, the 'process control cycle' as taught by Walter Shewhart of AT&T from the 1920s onwards and, by his pupils, W. Edwards Deming from the late 1940s to the 1990s (Deming 1986) and Joseph Juran (1974). It is one of nature's great laws; learn, adapt and survive.

Evo expands on the Statistical Process Control 'Plan-Do-Study-Act' (PDSA) cycle concepts since it demands:

[1] In 1988, the author taught Evo to an HP project team, which included Todd Cotton, who later went on the spread the method widely at HP (Cotton 1996), (May and Zimmer 1996).

- *Early* delivery of project results to stakeholders (for example, 'next week'!)
- *Frequent* releases to stakeholders (for example, 'every Friday')
- *Small* increments ('steps') (for example, no more than 2% of total project)
- *Useful*-to-stakeholder steps (benefit delivered, value experienced)
- Selection and sequencing of steps according to degree of stakeholder benefit; usually but not always, high-profit steps first (using dynamic priority determination).

Who could be against such an idea? It is a powerful competitive weapon. In practice, the main problem for project management is usually 'how?' How is a major project divided up into a succession of say, monthly improvements to be delivered into the hands of the users? Some people don't see any difficulty. Many, however, are unable to envision such small step decomposition for their projects, and usually claim it is impossible. In my experience, there are *always* ways of achieving such decomposition. It is a question of training, being determined to find the answer and having the right technical knowledge and/or sufficient insights into the stakeholder environment.

As our entire political, technological and economic world now has a greater rate of change and is much more unpredictable and complex than ever before, adaptive methods, such as Evo, must

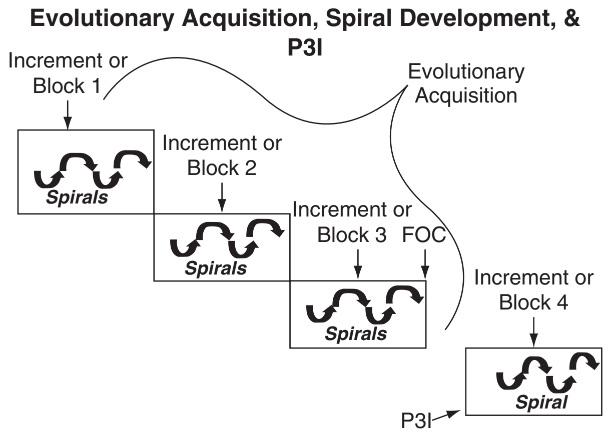


Figure 10.1
Illustration from letter to senior staff in the US Department of Defense from Under Secretary of Defense E. C. Aldridge, Jr., April 12, 2002. ''Since the publication of DoD Directive 5000.1 and DoD Instruction 5000.2, in which the Department established a preference for the use of evolutionary acquisition strategies...'' See http://www.acq.osd.mil/dpap/Docs/ar/1_multipart_xF8FF_2_EA%20SD%20Definitions%20final.pdf. This illustration is included mainly to show that evolutionary methods have been accepted at top levels within US Government.

become the norm to manage projects, rather than the exception. (It is only in the case of predictable, low-risk, low-turbulence, low-competition situations that there is little need for an evolutionary method.)

A Basic Evolutionary Planning Policy

1. **Financial Control**: No project cycle can exceed 2% of total initial financial budget before delivering some measurable, required results to stakeholders.
2. **Deadline Control**: No project cycle can exceed 2% of total project time (For example, one week for a one year project) before delivering some measurable, required results to stakeholders.
3. **Value Control**: The next step should always be the one that delivers best stakeholder value for its costs.

Policy Example: Formulating an Evo policy is the first stage of deciding how to do Evo. I frequently recommend this Evo policy to senior managers who must implement and support an Evo project management policy. You can adjust the level of detail to suit your environment.

10.2 Practical Example: Evolutionary Project Management

I have used this following example on numerous occasions. It stands such repetition as it gives such good insight into one common perceived barrier to Evo: "Delivery cannot be done until the new thing (such as a building, organization or IT system) is *ready* in some years time."

The Naval Radar System

Once, when holding a public course on Evo in London, a participant came to me in the first break and said he did not think he could use this early-incremental method. Why? "Because my system is contracted to be mounted on a new ship, not destined to be launched for three years".

I did not know anything about his system, at that point. But I expressed my confidence that there is *always* a solution to making a project evolutionary. So, I 'bet' that we could find an Evo solution during our lunch break. He sportingly accepted.

At lunch, he started by explaining that his research team made a radar device that had two antennas instead of the usual one (the dual signal sources were analyzed by a computer, which presented their data). It was for monitoring the ship- and air-traffic surrounding the ship it was on. This, I understood, was similar to having two eyes, instead of being a cyclops.

I then made a stab at identifying the 'results' he was delivering and who his stakeholders were (two vital insights for making Evo plans). "May I assume that the main result you provide is 'increased accuracy of perception', not just a black box, and that your major stakeholder is 'The Royal Navy', not primarily the ship (also one of the many stakeholders) itself?" "Correct", he replied. (I'm simplifying a bit, but the point to note is that identifying the primary real requirements and stakeholders, gives a 'wider playing field'.)

"Does your 'black box' work, more or less, now, in your *labs* (another stakeholder)?" I ventured. (Because, if it did, that opened for early use of some kind.) "Yes", he replied. "*Then what is to prevent you from putting it aboard one of Her Majesty's current ships* (yet other stakeholders!)? Initially running in parallel with conventional radar. Then ironing out any problems in practice. Enhancing it. Possibly giving that ship itself immediate increased capability, in a potential sudden real war? Then, when your new ship is launched, your system will be far more mature and safe to use", I tried innocently. (Actually, he got these points before I said anything!)

"*Nothing*!" he replied. And at that point I had won my bet, 20 minutes into the lunch.

"You know, Tom", he said after five minutes of silent contemplation, "the thing that really amazes me, is that not one person at our research labs has ever dared to express such a thought!"

Notice the 'method' emerging from this example:

1. Identify the primary stakeholders. Do not get distracted by secondary stakeholders. The primary stakeholder was not the 'new ship'. It was the 'Royal Navy' or even 'The Western Alliance.'
2. Look for the primary and real performance objective. Do not get distracted by the *perceived* project 'product' or secondary and supporting requirements or designs (like the radar system with two sources). Keep asking 'why?' until you find the primary objectives.

   The real objective was not 'to put the electronics box on the new ship'. It was 'an increased accuracy of perception.' In other words, 'an improvement in a performance aspect ("perception": a quality) of the radar function.'

The moment you have converted the result into such a 'scalar requirement,' then *evolving* towards your targets along that scale seems relatively easy to plan. This is one reason why we emphasize quantifying performance requirements in Planguage for use in Evo.

3. *Focus* your activity on delivering the required results to the stakeholders. Plan to deliver the results in increments or 'steps' to stakeholders. One reason is in order to get feedback from stakeholders on the way. There are several practical tactics you can use for identifying potential Evo steps, see Figure 10.6.
4. Don't take the formal contract too literally! *Early useful* results are *always* welcome, even when not demanded in a contract or requirements. (Check! Does the contract or requirements specification actually say: "We refuse to accept early delivery of any useful, partial results"?)
5. Don't assume that your project staff is thinking along these lines. Evo is not yet 'normal thinking' even amongst well-trained engineers.
6. Avoid Narrow Distractions. Think Big. Think System-wide.

10.3 Language Core: Evolutionary Step Specification

Step Content

An evolutionary step ('step' or 'Evo step') is a package of one or more design ideas.

A step has to be capable of being delivered as an organic whole. A step should stand 'on its own' as a complete deliverable.

A step also will have a defined size constraint. From the point of view of risk control, any step should only consume between 2% and 5% of the total project budget for time and financial cost. This has implications for the decomposition of design ideas, function changes and the scope covered by a step.

Delivery of a step is always intended in some defined way to move a real system (very occasionally, a trial system) in *the direction of its specified requirements*. A step might modify a system's function attributes, its performance attributes and/or its resource attributes. (Of course, when implemented, a step might not produce the expected results. It could, for example, have unintended, unexpected and undesired side effects.)

At some stage during the design process (maybe when initially specified, maybe later), a step is identified for delivery at some specific time, place and on some defined event conditions for a system. The conditions are specified using *qualifiers (for example, [Europe, Next Year, If a Competitive Product is on the Market].* 'Place' qualifiers can be any useful combination of system users, system locations, system components and system functions (*for example, [{Marketing Staff, Accountants}, {Europe, North America}, {Software Products, Training Products}, {Accounting, Marketing, Analysing Monthly Reports}]*).

Step Name

For communication purposes, a step may be named after its dominant content. A unique step-reference name is useful for specification of reuse of a step. Step names can have qualifiers (for example, Step A [Europe]).

EXAMPLE

Web Plan [Europe]:
Type: Evo Plan.
Consists Of: Step {Handbook, On-line Help, French Language Variant, Remove Spelling Mistakes, Provide For Customer Feedback}.

Step Dependency

Delivery of some steps might be dependent on other steps having already been delivered. Any step dependency must be explicitly stated in a step specification. In some cases, step dependency will be total and it will be impossible to deliver unless the dependency is met. In others, the step will be *capable* of delivery, but its impact on the requirements, such as meeting goals, will be significantly less.

EXAMPLE

Web Plan [Europe]:
Type: Evo Plan.
Consists Of: Step {Handbook, On-line Help, French Language Variant, Remove Spelling Mistakes, Provide for Customer Feedback}.
Dependency: Remove Spelling Mistakes Before Provide for Customer Feedback.

Step Sequencing

An Evo plan is a set of sequenced and/or a set of yet-to-be-sequenced steps. The current planned sequence of delivery of any of the steps should be reconsidered after each step has actually been delivered and the feedback has been analyzed. Many factors, internal and external, can cause a re-sequencing of steps and/or the insertion of previously unplanned additional step(s) and/or the deletion of some step(s).

It is the identification of the *next* step for delivery that should be our focus for detailed practical planning. After all, at the extreme, other planned steps may never be implemented in practice. So, the minimum information, initially needed for a step, is that needed to support the decisions for step sequencing.

When determining step sequencing, there are several factors including:

1. *Step dependencies* with other potential steps.
2. The *value that stakeholders will obtain if a step is delivered.* This is the key factor. Ask your stakeholders what unfulfilled requirements would be of most value to them and ask them for evidence supporting their choices. Different stakeholders might well choose different requirements. It could be that a requirement 'wins' due to its aggregated value to several stakeholders.
3. The *value to cost ratios* and the *performance to cost ratios* for steps (generally, the step with the highest value to cost ratio is implemented *earliest*. The performance to cost ratios are also another consideration).
4. The *gaps* between benchmark levels, or currently delivered levels, and the goals ('the biggest' gap or 'the toughest' gap is highest priority). The requirement with the biggest gap is the requirement that is least satisfied (that is, the lowest percentage of the way from the baseline towards the target has been achieved). A requirement, which is considered very tough, might also be a priority to start work on.
5. Stakeholder *opinion* (which can overrule any other logic). "I want this now!"

Specification using Planguage

There are many ways to express Evo plans. You can use any style that suits you. Here are some examples showing how Evo plans and steps can be specified using Planguage.

EXAMPLE Early Adopters:
Type: Step [Base = Current Product].
Consists Of: {DIF: Function [Country = USA]: Get Geographical Data, DIA, DIB}.

Note:

i) DIA, DIB and Get Geographical Data will have been previously specified and defined elsewhere. The design idea, DIF is defined here.
ii) 'Early Adopters' is a step, which consists of a package of three design ideas, DIF, DIA and DIB.

iii) Future steps are incremental additions to an existing system. We can define the system prior to delivery of a step by using the parameter 'Base'.

iv) In the 'Early Adopters' step, we explicitly declare the specification to be 'Type': (<- that's the 'explicit' part) 'Step.' 'Function' is used to tell the reader the type of Get Geographical Data.

v) The function Get Geographical Data is limited to one specific Country, USA.

Here is an example of stating an Evo plan.

EXAMPLE Product Plan:
Type: Evo Plan.
Includes: Step {SP [Before Rest], Rest: {SM, Step = SV, SX [After SM], SZ}}.

This example shows that an entire Evo plan can be summarized, at a high level, in a single Planguage statement. An Evo plan consists of a series of steps. In this example, there is a set of steps {SP, SM, SV, SX, SZ} which make up the Evo plan, named 'Product Plan.' 'Rest' is defined as being part of an Evo plan and consists of four defined steps. 'SP' is defined as the step to do 'before' the steps in Rest. The other steps have not had their sequence determined. There are no restrictions on doing them in parallel, or in arbitrarily convenient sequences, except that SX must be done 'after' SM.

Qualifiers specify 'where' a step is to be implemented. Here are two examples showing system location and system function being stated:

EXAMPLE Step 22:
Type: Step.
Consists Of: SR [State = {CA, NV, WA}].
'Step22' implements 'SR' in three US states: CA, NV and WA.

EXAMPLE Step 1:
Type: Step.
Consists Of: SP [Country = North America = {USA, CAN, MEX}, FX [State = {WA, GA, FL}]].

'Step 1' implements 'SP' in three countries. The function, FX within SP, is however restricted to just three US states. Note, the geographical concept or market area, 'North America' is defined here locally, for intelligibility or future reuse.

10.4 Rules: Evolutionary Project Management

Tag: Rules.EVO.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: Rules for Evo Plan Specification.

Base: The rules for generic specification, Rules.GS apply as well as all other Planguage rules needed to express requirements and design.

R1: **Tags**: All steps of an Evo plan will have a unique tag to enable cross-referencing from other specifications (such as test planning or costing).

R2: **Detail**: All detailed design idea specifications shall be kept separate from the Evo plan. For brevity, use Planguage step descriptions only. Any Evo plan elements yet to be defined in detail must be specified by a unique tag in fuzzy brackets (<Tag Name 1>). This will indicate that the detail is not specified yet. *Rationale: We need to avoid the clutter of design idea definitions in the Evo plan itself. Tags are sufficient.*

R3: **Cost**: Any planned step, that has an estimated incremental impact, for any resource attribute, which exceeds 5% of the total budget planned level, will be re-specified into smaller steps, to reduce risk. An average of 2%-of-budget steps is desirable (*as risk of economic loss is then at 2% maximum*), but individual projects *may* specify their own budget constraints. All planned steps still exceeding these single step budget constraints must be agreed by authorized signature.

R4: **Time**: Any step, which would take more than 5% of the total project calendar time (from project start up to the main long-term deadline), must be divided into smaller steps. An average of 2%-of-time steps is desirable, but individual projects may specify their own time constraints. All steps exceeding the 5% time constraint must be agreed by authorized signature. *Rationale: Control time to deadline.*

R5: **Priority**: The 'next step', at any point in the project, should ideally be selected using an Impact Estimation table to evaluate step options. Steps that you estimate to deliver the greatest stakeholder benefits, performance improvements (Sum of Percentage Impacts) to stakeholders, or that have the best performance to cost ratio, shall generally be done earliest, wherever logically possible, and when 'other considerations' (such as a customer contract or request) do not have higher priority. Any specific priority factors, which override going for the greatest stakeholder benefits first, shall be clearly documented.

There must be some *specified* clear rationale, policy or rule behind prioritizing steps differently from this rule. This could be some estimate of value of a step, which is outside the scope of the specific Impact Estimation table, which might have priority.

EXAMPLE Step 44:
Type: Step.
Consists Of: ABC [UK]: <- Contract Requirement 6.4.
Rationale: The contract demands we deliver this step at this point.

Optionally, there can be a project-defined constraint of a step having to achieve a minimum estimated value (financial growth or saving), overall performance improvement or performance to cost ratio before being considered for implementation at all.

R6: **Next**: Only the current step, or the approved next step, has 'commitment to implementation' (and even then, it could be terminated mid-implementation, if seen not to be delivering to plan). The sequencing specification of subsequent steps is not necessary and is certainly not fixed. *In practice, there is likely to be a tentative step sequencing mapped out, which captures any dependencies.*

R7: **Impact**: The next step must be numerically estimated *in detail* for its impacts on all the critical performance and resource requirements. Other later steps may be more roughly estimated, either individually or in relevant groups. *They will be estimated in greater detail as their 'turn' approaches. Rationale: To force us to estimate, measure and consider deviation in small immediate steps.*

R8: **Learn**: The actual results of the steps already implemented (that is, the cumulative impacts on all requirement levels to date) and the estimated results for the next step must be specified in an IE table (see Table 10.1 example). Specific comment about negative deviations already experienced, and what you have specifically done in your plan to learn from them, should be included in some form of footnote or comment. *(Note: We assume the use of an IE table, but other formats are possible.)*

R9: **Completeness**: *All* the specified design ideas for a system, implemented or not, must be represented *somewhere* on an Evo plan. (Remember, you can use tags and you can declare a large set of designs with a single tag. For example, A: Defined As: {B, C. D, E, F}.)

Rationale: This is because failure to include all the specified design ideas somewhere on the Evo plan causes confusion. It leaves us to wonder:

- *Was it forgotten inadvertently?*
- *Why is it specified, if it is planned never to be implemented? (If you are just keeping the idea in reserve, be specific.)*

10.5 Process Description: Evolutionary Project Management

These Evo processes are generalized. Modification to suit individual circumstances might well be required.

See also Figures 1.3 and 1.7 in Chapter 1.

Process: Strategic Management Cycle ('The Head')

Tag: Process.SM.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Note: Process.DC (Delivery Cycle) is a separate process defined below.

Entry Conditions: Entry.SM

E1: All necessary input information for Evo is available to the project management and design team.

E2: All input documents have successfully exited from their own quality control process. The specification quality control (SQC) entry condition applies to the project requirements and the design idea specifications. *Note: This usually implies between 0.2 and 1 remaining major defect(s)/page (A page is 300 words of non-commentary text.)*

E3: The design idea specifications have been evaluated using IE and, the IE table has exited from SQC.

E4: The level of uncertainty acceptable to the project has been formally determined (deviation (± %) from plan). Default level ± 10%.

E5: The project management and design team are adequately trained or, assisted by a qualified person to analyze and specify evolutionary plans.

E6: There is relevant approval, including funding, for the project to proceed.

Procedure: Procedure.SM

P1: Plan:

1. Modify if necessary top-level project requirements and design ideas.

2. Update the long-term Evo plan.
3. Initiate any backroom development cycles and/or production cycles required for future steps.
4. Decide on the next step for delivery (to the frontroom).
5. For next step: Set *step* targets, select *step* design ideas, decide *step* [qualifiers].
6. Produce maximum one page overview plan for the step delivery (see template in Figure 10.8 and, also the example in Figure 10.5).

The step delivery cycle (DC) can start once the next step (for delivery) has been decided and when the relevant development and production cycles are complete.

P2: Do:

Initiate the Delivery Cycle (that is, the step delivery to the stakeholder. Others may carry out the detailed work).

P3: Study:

1. *On completion* of the Delivery Cycle, identify the numeric differences between the system's actual attribute levels and the target requirements. Where are the large 'gaps'?
2. Note numeric differences between *estimated* step results and *actual* results.
3. Monitor the progress of any current 'backroom' development cycles and/or production cycles. Ensure they have sufficient resources to be completed on-time.
4. Note any stakeholder needs, technological, political or economic changes, which should be reflected in the Evo step sequencing, or even the requirement or design specification.

P4: Act:

Adopt the change, or *abandon* it (revert to previous state before step implementation). Or, decide to run through the cycle again, but possibly under changed conditions *(paraphrased from W.E. Deming 1986).*

Go to P1 (that is, *continue cycling*), unless Exit Conditions are met.

Exit Conditions: Exit.SM

X1: If resources used up, stop project. Keep results achieved so far!

X2: If all existing Goal levels are reached, stop using resources.

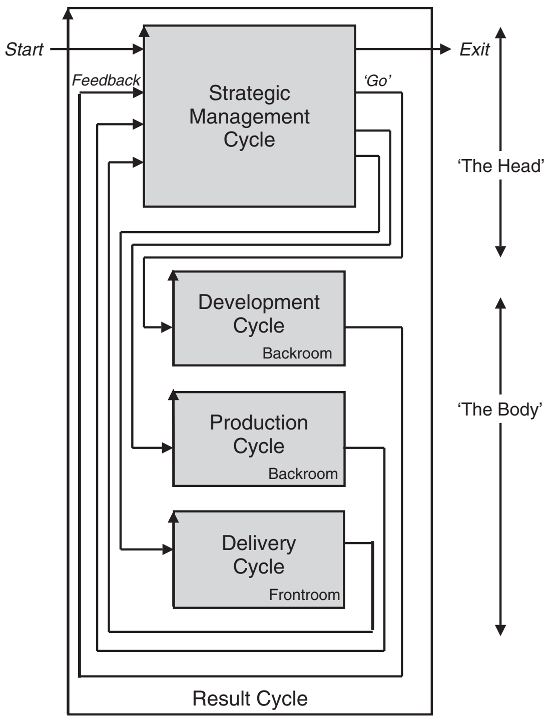


Figure 10.2
The result cycle for an Evo Step.

Process: Delivery Cycle (Part of 'The Body')

Tag: Process.DC.

Version: October 7, 2004.

Owner: TG.

Status: Draft.

Gist: This process is for delivery of a single step, not the larger project totality.

Entry Conditions: Entry.DC [Step n]

E1: All logically prerequisite steps to this one, which were specified, have been completed.

E2: The numeric feedback results from any previously completed steps must be available to the design team and must have been studied. *(You may want to re-do the previous step before proceeding.)*

Procedure: Procedure.DC [Step n]

P1: Plan:

1. Specify the delivery of the step in detail. See Figure 10.8 in Section 10.9 for a template.
2. Agree the plan with the relevant, affected stakeholders (For example, management and customers). The list of topics to consider includes: changes to working practices, training, installing, regression testing, field trials, hand-over and criteria for success: all system-wide considerations.

P2: Do:

Deliver the step. Install it with real stakeholders, so they get some of the planned measurable benefits.

P3: Study:

1. Determine the results of delivering the step. Obtain any relevant measurements: test, measure and sample, to establish the new performance levels and the new operational cost levels. Compare results to the short-term and long-term targets.
2. Analyze the data and produce a feedback report for management.

For example, use an Impact Estimation table as a tool to do this study task.

P4: Act:

1. Decide if this step succeeded, must be redone in whole or part, or totally rejected.
2. Take any required minor corrective actions (for example, bug-fixing) to 'stabilize' the system.

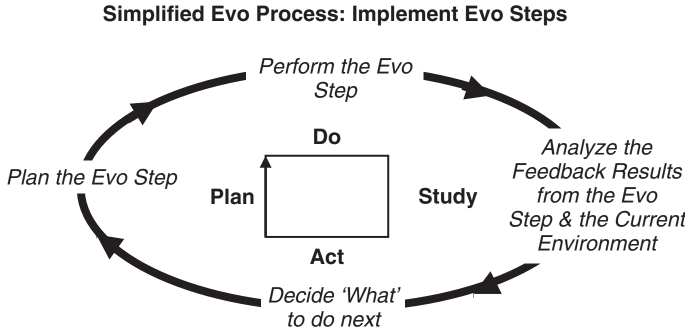

Simplified Evo Process: Implement Evo Steps
Perform the Evo Step
Do
Plan the Evo Step
Plan
Study
Analyze the Feedback Results from the Evo Step & the Current Environment
Act
Decide 'What' to do next

Figure 10.3
A simplified Evo process: implementing Evo steps.

Exit Conditions: Exit.DC [Step n]

X1: Step completed, or dropped. Exit a step only when all step performance levels and function requirements are reached (or wavered formally). Give up if reaching planned requirements is impractical, or if you run out of resources.

Note: Process Descriptions for the Development Cycle and the Production Cycle are not given in this text.

A Simplified Evo Process

Background: A simplified version of the Evo process to use on small projects. It also serves to help understand the larger, full-scale Evo process.
Tag: Simplified Evo.
Version: October 7, 2004.
Owner: TG.
Status: Draft.

Process Description

1. Gather from all the key stakeholders the top few (5 to 20) most critical goals that the project needs to deliver. Give each goal a reference name (a tag).
2. For each goal, define a scale of measure and a 'final' goal level. For example: *Reliable: Scale: Mean Time Before Failure, Goal: >1 month*.
3. Define approximately 4 budgets for your most limited resources (for example, time, people, money and equipment).
4. Write up these plans for the goals and budgets (*try to ensure this is kept to only one page*).
5. Negotiate with the key stakeholders to formally agree the goals and budgets.
6. Plan to deliver some benefit (that is, progress towards the goals) in *weekly* (or shorter) increments (Evo steps).
7. Implement the project in Evo steps. Report to project sponsors after each Evo step (weekly, or shorter) with your best available estimates or measures, for each performance goal and each resource budget. *On a single page*, summarize the *progress to date* towards achieving the goals and the costs incurred.

Policy

- The project manager and the project will be judged exclusively on the relationship of progress towards achieving the goals versus the amounts of the budgets used. The project team will do anything legal and ethical to deliver the goal levels within the budgets.
- The team will be paid and rewarded for benefits delivered in relation to cost.
- The team will find their own work process and their own design.
- As experience dictates, the team will be free to suggest to the project sponsors (stakeholders) adjustments to 'more realistic levels' of the goals and budgets.

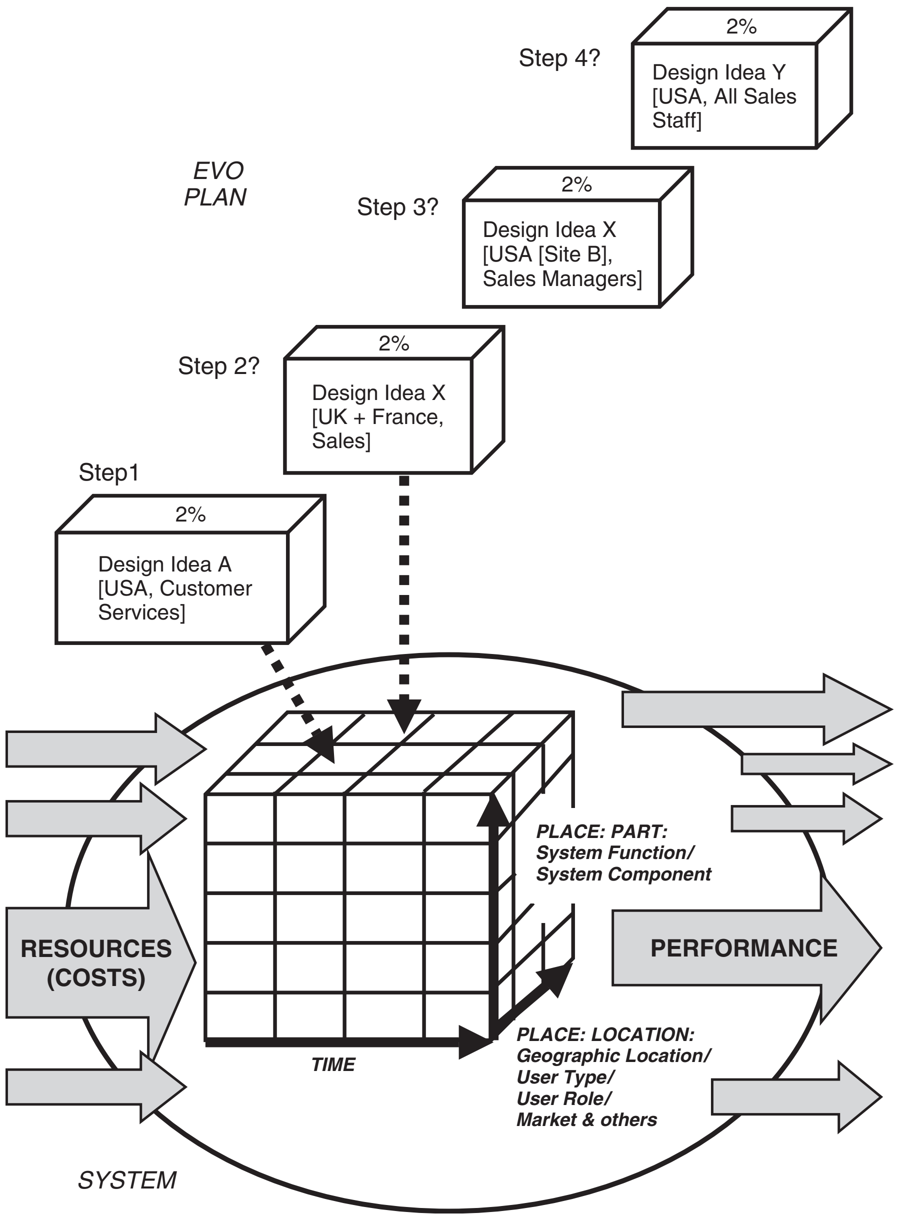

Step 4?
2%
Design Idea Y [USA, All Sales Staff]
EVO PLAN
Step 3?
2%
Design Idea X [USA [Site B], Sales Managers]
Step 2?
2%
Design Idea X [UK + France, Sales]
Step1
2%
Design Idea A [USA, Customer Services]
RESOURCES (COSTS)
PLACE: PART: System Function/ System Component
PERFORMANCE
TIME
PLACE: LOCATION: Geographic Location/ User Type/ User Role/ Market & others
SYSTEM

Figure 10.4
An Evo plan and the system: the diagram shows the steps being sequenced for delivery. Each step delivers a set of performance attributes (a subset of the long-term planned results), and consumes a set of resources (a subset of the long-term budgets), in a specific place {location, system component} and at a specific time (for delivering the benefits). The purpose of this diagram is to show that each Evo Step will become a sub-component of the evolving system's long-term vision and plan.

10.6 Principles: Evolutionary Project Management

1. **The Principle of 'Capablanca's next move'**
   There is only one move that really counts, the next one.
2. **The Principle of 'Do the juicy bits first'**
   Do whatever gives the biggest gains. Don't let the other stuff distract you!
3. **The Principle of 'Better the devil you know'**
   Successful visionaries start from where they *are*, what they *have* and what their *customers* have.
4. **The Principle of 'You eat an elephant one bite at a time'**
   System stakeholders need to digest new systems in small increments.
5. **The Principle of 'Cause and Effect'**
   If you change in small stages, the causes of effects are clearer and easier to correct.
6. **The Principle of 'The early bird catches the worm'**
   Your customers will be happier with an early long-term stream of their priority improvements, than years of promises, culminating in late disaster.
7. **The Principle of 'Strike early, while the iron is still hot'**
   Install small steps quickly with people who are most interested and motivated.
8. **The Principle of 'A bird in the hand is worth two in the bush'**
   Your next step should give the best result you can get now.
9. **The Principle of 'No plan survives first contact with the enemy'**[2]
   A little practical experience beats a lot of committee meetings.
10. **The Principle of 'Adaptive Architecture'**
    Since you cannot be sure where or when you are going, your first priority is to equip yourself to go almost anywhere, anytime.

[2] This saying is attributed to Prussian general staff and the elder Von Moltke: "They did not expect a plan of operations to survive beyond the first contact with the enemy. They set only the broadest of objectives and emphasized seizing unforeseen opportunities as they arose . . . Strategy was not a lengthy action plan. It was the evolution of a central idea through continually changing circumstances" (From Von Clausewitz in his 'On War', quoted by General Electric's CEO, Jack Welch in a speech December 8, 1981, in Slater, 2000: 194).

The Principles of Tao Teh Ching (500 BC)

That which remains quiet, is easy to handle.
That which is not yet developed is easy to manage.
That which is weak is easy to control.
That which is still small is easy to direct.
Deal with little troubles before they become big.
Attend to little problems before they get out of hand.
For the largest tree was once a sprout, the tallest tower started with the first brick, and the longest journey started with the first step.[3]

10.7 Additional Ideas: Evolutionary Project Management

Backroom/Frontroom

Some step components will inevitably have a longer development and/ or production elapsed time. This can be due to a variety of reasons, for example, lead time for purchasing. In such cases, 'backroom' activities will have to be underway well before the decision about which step to deliver next is made. There will have to be parallel step component development and production cycles.

The stakeholder in the frontroom, who receives the delivery step, is unaware of the backroom work. A useful analogy is a restaurant kitchen (backroom) and the customers in the restaurant (frontroom): if all goes well, the food is delivered at frequent intervals and all the preparation, cooking time and co-ordination of dishes for a specific table are invisible to the customers.

In fact, a skillful project manager will probably aim to have more than one potential delivery 'stockpiled,' so that there is choice over the next delivery, and leeway if any major problem effects step development.

Using IE Tables for Evo Plans

The steps of an Evo plan can be analyzed using an IE table. See Table 10.1. In this case steps (which consist of design ideas) are specified. Steps are estimated and (after deployment) measured for impact on requirements (performance and resource attributes), in relation to targets.

[3] From Lao Tzu, in Bahn (1980).

Table 10.1 This is a conceptual example. Three goals (performance targets) and two resource targets are having the real impacts on them tracked, as steps are delivered. The same IE table is also being used to specify the impact estimates for the future planned steps. So at each step, the project can learn from the reality of the step's deviation from its estimates. Plans and estimates can then be adjusted and improved from an early stage of the project.

Step	*Step 1*			*Step 2 to Step 20*		*Step 21 [CA, NV, WA]*		*Step 22 [all others]*	
Target Requirement	Plan % (of Target)	Actual %	Deviation %	Plan %	Plan % cumulated to here	Plan %	Plan % cumulated to here	Plan %	Plan % cumulated to here
Performance 1	5	3	−2	40	43	40	83	−20	63
Performance 2	10	12	+2	50	62	30	92	60	152
Performance 3	20	13	−7	20	33	20	53	30	83
Cost A	1	3	+2	25	28	10	38	20	58
Cost B	4	6	+2	38	44	0	44	5	49

An Evo Step Specification

Evo Step: Tutorial [Model 1234, Basic].

Stakeholders: {Marketing, Department XX}.
Implementers: Department XX.
Intended Audience: Marketing.

Gist: To prepare a written tutorial that teaches how to identify required information on internet web pages.

Step Content: HCTD12: <Hard Copy Text Document>. "This declares a design idea, HCTD12, that needs further detailed specification. Some additional notes about it are also given. See below."

Notes [HCTD12]:

- Can write the basic minimal functions, MMM, in 1 week. <-GF.
- Provide step by step instructions, in English.
- Questionnaire for Stakeholders.
- Intended audience: Marketing.
- Focus on <sales aspects>, not how to identify information in detail (not yet, in this step).
- Go to <specific web sites>.
- Process for Testing with Stakeholder (for example, observation, times).
- Pinpoint some characteristics of what we see on the terminal compared with what we see on a <PC or other terminal>.
- What instructions should be on the terminal to begin?
- No illustrations to be provided, just text.

Questionnaire: Defined As: Questionnaire to walkthrough with stakeholders.

Step Validation: Defined As: Process for Testing with Stakeholders. "Example observation, times."

Constraint: Step must be deliverable within one calendar week.

Assumptions [Applies = Step Cost [Effort], Source = MMM]: 10 hours per page.

Dependencies: <Feature list of WWW>, <77777 WWW Browser> <-MMM.

Risks: At least 3 hours needed of TTT's time for input and trial feedback.

Step Value:
{[Stakeholder = TTT, Saleability]: <some possibility of value>,
[Stakeholder = Developers]: <value of feedback on a tutorial>}.

Step Cost [Effort]: < 10 hours <-MMM.

Figure 10.5
An example of using the specification template for an Evo step.

Notes:

1. *New user-defined types of 'Questionnaire' and 'Intended Audience' have been locally declared.*
2. *There is a brainstorming and note-taking atmosphere here. Do not expect to understand the internal language of my client in an isolated first draft teamwork example!*
3. *This illustration is mainly given to show an example of a set of parameters describing the attributes of a step. You should feel free to design your own set of useful step specification parameters.*

Using Templates for Specifying Evo Steps

Figure 10.5 shows an example of a filled-in template for specifying an Evo step (*see also Figure 10.8 in Section 10.9 for an outline template*).

How to decompose systems into small evolutionary steps: (a list of practical tips)

1. Believe there is a way to do it, you just have not found it yet![4]
2. Identify obstacles, but don't use them as excuses: use your imagination to get rid of them!
3. Focus on some usefulness for the stakeholders: users, salesperson, installer, testers or customer. However small the positive contribution, something is better than nothing.
4. Do *not* focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!
5. Think one stakeholder. Think 'tomorrow' or 'next week.' Think of one interesting improvement.
6. Focus on the results. (You should have them defined in your targets. Focus on moving *towards* the goal and budget levels.)
7. Don't be afraid to use temporary-scaffolding designs. Their cost must be seen in the light of the value of making some progress, and getting practical experience.
8. Don't be worried that your design is inelegant; it is results that count, not style.
9. Don't be afraid that the stakeholders won't like it. If you are focusing on the results they want, then by definition, they should like it. If you are not, then do!
10. Don't get so worried about "what might happen afterwards" that you can make no practical progress.
11. You cannot foresee everything. Don't even think about it!
12. If you focus on helping your stakeholder in practice, now, where they really need it, you will be forgiven a lot of 'sins'!
13. You can understand things much better, by getting some practical experience (and removing some of your fears).
14. Do early cycles, on *willing local mature* parts of your user/stakeholder community.
15. When some cycles, like a purchase-order cycle, take a long time, initiate them early (in the 'Backroom'), and do other useful cycles while you wait.
16. If something seems to need to wait for 'the big new system', ask if you cannot usefully do it with the 'awful old system', so as to pilot it realistically, and perhaps alleviate some 'pain' in the old system.
17. If something seems too costly to buy, for limited initial use, see if you can negotiate some kind of 'pay as you really use' contract. Most suppliers would like to do this to get your patronage, and to avoid competitors making the same deal.
18. If you can't think of some useful small cycles, then talk directly with the real 'customer', stakeholders, or end user. They probably have dozens of suggestions.
19. Talk with end users and other stakeholders in any case, they have insights you need.
20. Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.

[4] Working within many varied technical cultures since 1960 I have never found an exception to this – there is always a way!

Figure 10.6
Ideas to assist identifying steps.

10.8 Further Example/Case Study: The German Telecommunications Company

Here is an example of another perceived barrier preventing use of the evolutionary method: "It is too late, we have already invested so much in the old way, that we just have to see it through."

At a large German telecommunications business, almost 1,000 software engineers had been working for three years on a major new world-market product. The hardware was ready, but the software was late. I was told by the Financial Director for the project that the next month, December, was the actual deadline for product delivery, but that their 40,000 node PERT chart (really!) estimated they had two or three years more software effort left. Corporate marketing management had given them one more year, until December next year. They had to deliver, or forget the whole market, which by that time would be taken over by competitors.

I suggested re-planning the project into smaller steps with critical increments first. They told me that this was unthinkable: the software was 'already written' and they claimed that only testing remained. They also had a rather long list of other reasons why evolutionary delivery would not work for them.

Using common sense, we worked out a basic evolutionary plan: we used a day to plan and a second day to sell the idea. We decided we ought to aim to deliver the small-system software first (there were 35 signed contracts for it and, none for the medium and large systems). Then we decided to select for Evo delivery the fundamental telephone services before any advanced complex stuff.

After moving through what seemed like seven management layers (there were probably only four) with "You must present this to my boss," we ended up in the office of Herr R., the Project Director. He thought it was all good common sense, and stared coldly at his (cowardly, cautious?) subordinates as he asked: "Can you do it this way?" When they gave assenting nods, he merely said, "Then do it!"

They did! On a return visit in the November of the next year, they told me that the small systems had been operating for over six weeks with several real customers and with no problems whatsoever. Note, three months *before* the impossible deadline!

The lessons to learn include:

1. You must *look* for Evo steps, don't assume that others have done so,
2. Evo steps are usually clear, simple and found by using product knowledge and stakeholder requirements, and
3. You have to get the right person to make the decision to 'go' with Evo, usually a senior manager, who is focused on delivering business benefit.

The product remained a major successful product on the market for many years. Herr R. correctly concluded that unless the organization changed its mode of thinking, the same type of project problem would continue to recur. So he took steps to improve the organization. Prevention is better than cure!

10.9 Diagrams/Icons: Evolutionary Project Management

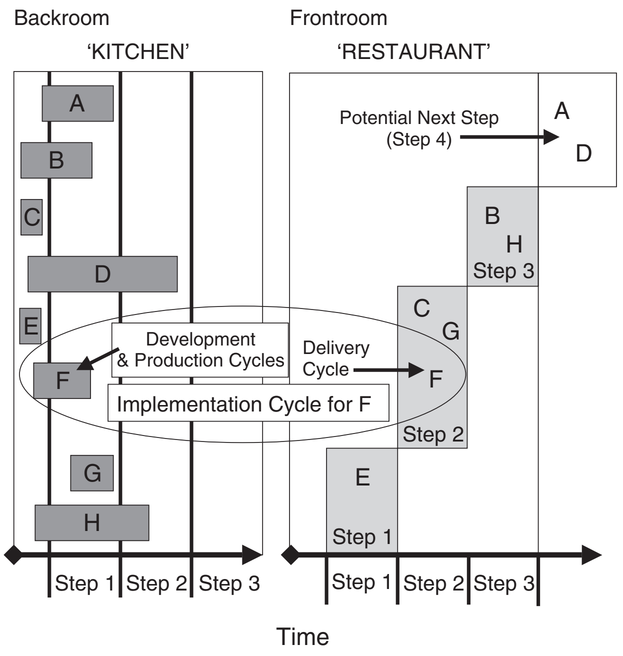


Figure 10.7
Backroom and frontroom activities: diagram showing the relationship between backroom and frontroom activities. The step components are developed and assembled in the backroom and then delivered in steps to the frontroom. A frequency of step delivery is maintained. Step 4 is actually ready ahead of its delivery time and is held back. Note, given when the system components were ready for delivery, there were several choices about the delivered step content. The step time lines in the backroom show when the corresponding frontroom steps were done. For example, G but not H, was complete by the beginning of Step 2, and G was therefore available for delivery if we decided to do that.

A Template For EVO Step Specification

Tag: <Tag name for the step>.
Type: Evo Step.
========================== Basic Information ==========================
Version: <Date or version of last update to step specification>.
Status: <{Specification Stage [{Draft, SQC Exited, Approved}], In Evo Plan, Scheduled Next, Under Implementation, Delivered awaiting Feedback, Feedback Obtained}, date> <- <Source (who says 'Status' is true?)>.
Quality Level: <Maximum remaining major defects/page, sample size, SQC date>.
Owner: <Who is taking responsibility for the step in terms of specification>.
Stakeholders: <Who are you going to deliver requirements to? >.
Implementers: <Who is in charge of implementing this step>.
Gist: <Brief description of the main idea of this step>.
Description: <Give a detailed, unambiguous description of the step, or a tag reference to a place where it is described. Remember to include definitions of any local terms>.

Implementation Details: "Includes relevant details, such as <which product>, <which area of application system>."
Evo Plan: <Tag of the Evo Plan that this step is associated with>.
Step Content: <Step Elements: {Design Ideas, Functions, Tasks, re-used step definitions}>.

=========================== Measurement ===========================
Test: <Refer to tags of any test plan and/or test cases, which apply to this step>.
Step Validation/Feedback:
Specification Quality Control (SQC): <outcome, date>,
Pre-Delivery Test: <outcome, date>,
Post Delivery Results: <{problems, stakeholder feedback}, date>,
Certification Specification: <refer to the certification plans>.

====================== Priority and Risk Management ======================
Constraints:
<Any legal, political, economic, security or other constraints imposed on implementation> <- <Source (who says this is true?)>.
Assumptions: <Any assumptions that have been made>.
Dependencies:
<Anything which must be in place, finished, working properly, for us to be able to start this Evo step or to complete it> <- <Source (who says this is true?)>.
Risks: <Any risks that need to be taken into account>.
Priority:
<Name, using tags, any system elements, which must clearly be done *after* or must clearly be done *before*. Give any relevant reasons>.
Issues: <Unresolved concerns or problems in the step specification or the system>.

========================== Benefits and Costs ==========================
Rationale: <Justify the existence of this step>.
Step Value:
<Real measurements or estimates of numeric value to stakeholders>. "Value in terms of meeting the requirements. At least, the value on scale 0 (none) to 9 (highest)."
<- <Source (who says this is true?)>.
Step Cost:
<Budgets or real costs>. "For example, financial costs and engineering hours. These must be constrained by the Evo 2% policy. At least, the value on scale 0 (very cheap) to 9 (high and unpredictable)." <- <Source (who says this is true?)>.

Figure 10.8
A possible specification template for a one-page Evo step. Notice that the parameters are designed to give you enough information to decide on the order for step sequencing in an Evo plan.

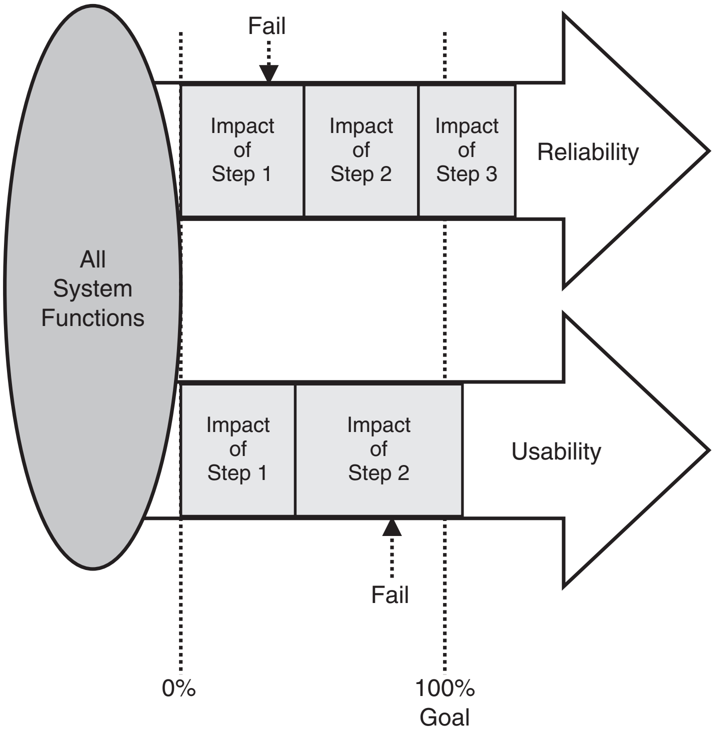


Figure 10.9
Dynamic priority: 0% is the baseline level – before we start further evolution of the system. After Step 1, usability has priority because it is not at an acceptable level (that is, not better than the specified fail level) yet. Reliability is above the fail level (and thus in the 'acceptable area'). After Step 2, reliability now has priority because it has not reached goal level yet. After Step 3, both reliability and usability have reached 100% of their respective goal targets. Consequently the project is 'finished' – no more performance gaps.

10.10 Summary: Evolutionary Project Management

Dr. Deming had a charming understated way of expressing the outcome of a venture: "Survival is not compulsory." Sadly, far too many projects demonstrate the truth of this and, in the process, waste years and large sums of money, and deliver nothing except weakened economy and reputation. Professor Peter Morris in his book, *The Management of Projects*, identifies that none of the existing well-known project management methods really enable sufficiently good control of projects (Morris 1994). He also outlines that the way forward must incorporate evolutionary methods. In fact, Evo exists and already has a track record of success; it is just not widely known and practiced within the systems engineering and other engineering communities.

Hopefully, this chapter has taught the basic concepts of Evo:

- Evo is, above all, the application of the Shewhart process control cycle, 'Plan-Do-Study-Act'. It is learning from doing and acting on that learning. It is adapting to the complex and changing realities of a project. Evo is systematic engineering work (of the type described by Koen (Koen 1984).
- Evo is primarily guided by well defined, quantified, *but not necessarily static*, multiple requirements for performance and cost. These requirements represent, at least indirectly, the value system of the stakeholders. Deviation from the path to reaching the requirements is corrected with minimum loss of resource. We are always open to corrections of our requirements. Such corrections are caused either because the world has changed or because we better understand how to formulate our 'values' in terms of requirements (Keeney 1992).
- Evo is concerned with *controlling risks*. By insisting on small steps, you learn early about the project's capability to deliver, and about the users, other stakeholders and their system environment. You are in a position to adapt to what you learn; and also to incorporate any additional changes requested.
- Evo demands early delivery of the high priority improvements. This gains credibility for the project and should attract resources to continue to do so.

Don't be fooled by the term 'evolutionary' into thinking Evo means 'slow and small change'. If you want change, even revolutionary change, then Evo project management:

- will give you better results
- will get you faster to market
- will help you meet your deadlines.

Planguage Concept Glossary

Glossary Introduction

Purpose of the Glossary

This glossary contains the master definitions of the fundamental Planguage concepts. Its central purpose is to define 'concepts' – not words. I view this concept glossary as a central contribution of this book, standing in its own right.

> "What's in a name? That which we call a rose, by any other name would smell as sweet."
>
> *Shakespeare, Romeo and Juliet, Act 2*
>
> "Every word or concept, clear as it may seem to be, has only a limited range of applicability."
>
> *Werner Heisenberg*[1]

With the Heisenberg quotation in mind, this glossary will try to give the interpretation Planguage intends, when the glossary terms are used in *this book*. (If the text and the glossary do not seem to agree, I suggest you trust the glossary primarily as a correct interpretation.[2])

Further explanation of the glossary-defined concepts is found in the main text (via the index). An updated and extended Planguage Glossary is also to be found on the website www.Gilb.com and at www.books.elsevier.com. Space limitations within the book meant that not all the glossary could be included.

Development of this Glossary

I have not tried to define *all* possible concepts for a systems engineering discipline. I have merely concentrated on defining those that I have found useful in my work.

Some other concepts have been included because the glossary has been developed in connection with drafting future books in this Planguage

[1] Heisenberg, Werner, 1958, *Physics and Philosophy*, London: Penguin Books (2000), ISBN 0-141-18215-6, 176 pages.

[2] I believe Bertrand Russell (1872–1970) said that if the experts disagree, you cannot be sure that either one of them is right. So, my advice to trust the glossary must be taken with caution!

series. (The intended titles are *Requirements Engineering, Priority Management* and *Evolutionary Project Management*. Unpublished versions and drafts of these are to be found on the website www.Gilb.com.) So, the glossary may seem somewhat detailed in the context of a single book. But, the intention is to have a common glossary across all the books in the series.

Development of Concepts

In defining a concept, I have not attempted to blindly follow any single particular standard, such as INCOSE, ISO or IEEE. Indeed, I regularly found them inadequate for the specialized purpose at hand. I have primarily tried to let the concepts suit my narrow 'systems engineering' purposes and, above all, to be consistent with each other.

It is worth explaining that I have had considerable help and feedback from my editor and a number of colleagues, correspondents, friends, students and clients regarding definitions, and the choice of primary terms. I have served as a final subjective decision-maker because in language there is no right or wrong, but it is central that the reader know what the writer intends.

I do view the glossary as open-ended for both my own purposes and for purposes of the reader. I also view each concept as potentially capable of continuous improvement in definition.

The Glossary as a Reader-Extendible Tool

I do not mean to impose my terms or definitions on the reader. I respect their rights and needs to define things, in any useful or traditional way for them. I also respect their right to rename any terms. I just needed to take a position on concepts and terms in order to communicate and develop my own ideas. I intend to develop the glossary as needed, and the reader should feel free to do the same, for their own uses and benefits.

The deeper I have gone into this glossary, the more humbled I have been with the infinite possibility of improvement. So, I beg the readers to accept the many imperfections as the best I could do within the timescales, and still publish it in book form at all. I promise to continue the improvement, to participate in improvements and to make this basis freely available at no cost or restriction to people who want to improve it or make it specialized for their own purposes. Permission is hereby given to quote from the glossary freely, and partially, provided suitable credit is given as to origin (© credit to Tom Gilb is sufficient). Notification of your use would always be interesting to me, and may result in useful updates and feedback to you. (Notification of use and reference to Tom@Gilb.com).

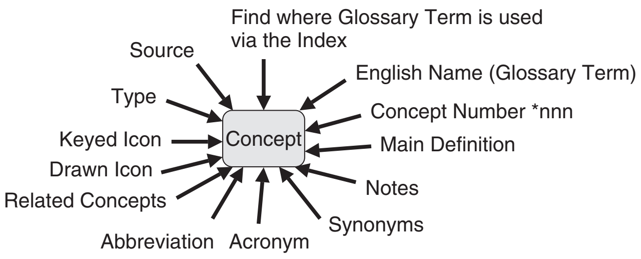


Figure G1

About the Glossary Concepts

A concept can have many 'names' (or 'tags' in Planguage), which act as pointers to it: the names do not change or determine a concept, but merely *cross-reference* it. The central, universal identification tag of a concept is its unique concept number, prefaced by an asterisk, *nnn (for example, *001). The idea behind the concept numbers is to allow and enable full or partial translation into various international languages and into corporate dialects.

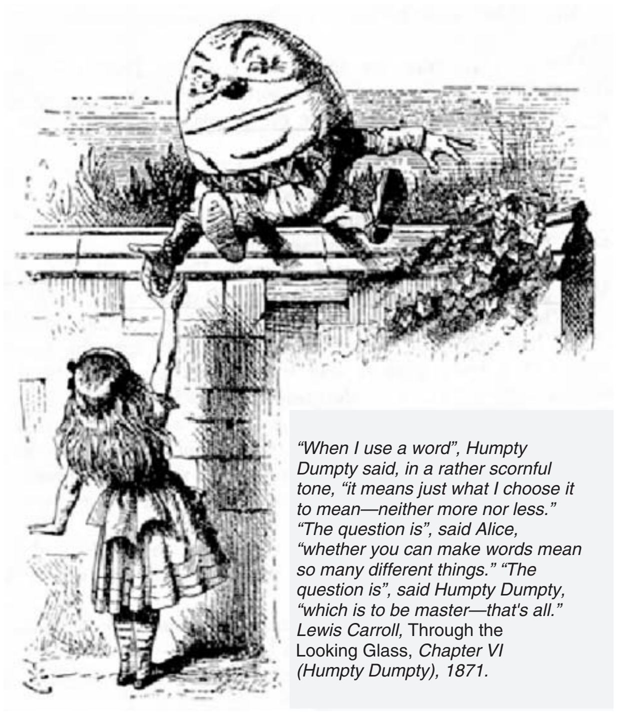

"When I use a word", Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean—neither more nor less." "The question is", said Alice, "whether you can make words mean so many different things." "The question is", said Humpty Dumpty, "which is to be master—that's all." Lewis Carroll, Through the Looking Glass, *Chapter VI (Humpty Dumpty), 1871.*

Figure G2
Alice meets Humpty Dumpty[3].

[3] Illustration by John Tenniel to Chapter 6 of 'Through the Looking-Glass' by Lewis Carroll. Wood-engraving by Thomas Dalziel. Illustration from http://www.scholars.nus.edu.sg/landow/Victorian/graphics/tenniel/lookingglass/6.1html/. Additional Lewis Carroll text was added in.

After **Concept *313**

'After' is used to indicate a planned sequencing of events, including Evo steps and tasks.

Example:
Product Trials: Step: {F2, F1 After F2}.
This means that Product Trials (a tag) is a defined Evo Step consisting of step elements F2, and then F1.

Aim **Concept *001**

An 'aim' is a stated desire to achieve something by certain stakeholders. An aim is usually specified informally and non-numerically.

Example:
Our aim is to be the dominant supplier of mobile phones in China by the end of the decade.
Note the two constraining qualifiers (China, By End of the Decade) and the function area (Supplying Mobile Phones).

"The aim must include plans for the future."
<-(*Deming 1993 Page 51*)

"It is important that an aim never be defined in terms of activity or methods. It must always relate to how life is better for everyone."
<-(*Deming 1993 Page 52*).

"The aim precedes the organizational system and those that work in it. Workers, for example, can not be the source of the aim, for how would one know what kind of workers to choose?"
<-(*Deming 1993 Page 52*)
Attributed to Deming by Carolyn Bailey.

Notes:
1. When using the term 'aim,' the intent may be to simplify, and give the ambition level.
2. An aim is ultimately specified in the complete and detailed requirements specification.

Example:
"Our aim is to have the <best> book on <gardening>."
Aim [New System X]: Superior long-term competitive edge in all market areas and product lines.

Related Concepts: Goal *109; Budget *480; Target *048; Ambition *423; Mission *097; Vision *422.

Ambition **Concept *423**

'Ambition' is a parameter, which can be used to summarize the ambition *level* of a performance or resource target requirement. Ambition must state the requirement concerned (like 'Usability') and it must contain a notion of the kind of level being sought (like 'high').

Notes:
1. The Ambition summary is useful for getting team understanding and agreement to its concept, before going on to the detailed specification

work. It can then be used during development of the specification as a basis for judging the relevance of the details. The Ambition can also be updated to reflect the detailed specification better, if desired.
2. Once the specification is completed, Ambition provides a useful overview summary of the more detailed specification.

Example:
Usability:
Ambition: The system will be extremely/competitively easy to learn, and to use, for a variety of users and user cultures.
Reference: Quality Attribute Usability Paper, Version 0.2.

Keyed Icon: @.Σ "Target and Summary."

And **Concept *045**

'And' is used as a logical operator to join any two expressions within a statement.

Example:
Goal [If War and Inflation]: 60%.
Goal [If Peace And Inflation]: 60%.
Goal [If War AND Stability]: 60%.
To make a statement read better, the lead capital letter can be dropped, giving 'and' rather than 'And'.

Architecture **Concept *192**

The 'architecture' is the set of components that exist in a system, and impact a set of system attributes directly, or indirectly, by constraining, or influencing, related engineering decisions.

Notes:
1. Interesting specializations:
   - **Perceivable Architecture**: the architecture, which is somehow directly or indirectly perceivable in a real system, as determining the range of performance and cost attributes possible. This applies regardless of who, if anyone, consciously specified the architecture design artefacts.
   - **Inherited Architecture**: the architecture, which was not consciously selected *for this system* at a particular level of architecture activity, but was either incidentally inherited from older systems or accidentally inherited from the specified design artifacts, specified by architects, managers or engineers.
   - **Specified Architecture**: the formally defined architecture specifications at a given level and lifecycle point, including stakeholder requirements interpretation, architecture specification, engineering specification done by this architecture level, certification criteria, cost estimates, models, prototypes, and any other artifact produced as a necessary consequence of fulfilling the architecting responsibility.
2. An extensive discussion of the architecture concept is given in (Maier 2002), including Appendix C on the history of attempts to define a standard within DoD, IEEE and INCOSE.

3. The highest specified level of design ideas for a defined system is called the 'architecture'. The architecture is the collection of controlling design ideas for a defined purpose. The architecture refers primarily to frameworks, interfaces and other technology and organizational ideas which more-detailed design ideas are expected to fit in to.
4. The architecture specifications (*617) would probably be classified as generic design constraints (or 'architecture constraints', if you wanted to emphasize the idea of 'architecture').
5. Architecture specifications would have priority over subsequent design decisions, made at more-specialized engineering levels.
6. 'Architecture specification' is the set of system-wide decisions, which are made in order to improve the systems survival ability, as it is threatened by changes to it, and by its environment.

Architecture: A high level design that provides decisions about:

- purpose (What problem(s) that the product(s) will solve)
- function description(s) (Why has it been decomposed into these components?)
- relationships between components (How do components relate in space and time?)
- dynamic interplay description (How is control passed between and among components?)
- flows (How does data or in-process product flow in space and time?)
- resources (What resources are consumed where, in the process or system?).

Source: Standard: FAA-iCMM Appraisal Method Version 1.0 A-19, INCOSE Conference CD, June 1999, Brighton UK [FAA98]

This definition differs from Planguage in that we are primarily concerned with design aspects, and this contains three requirement notions.

Architecture: The organizational structure of a system or component.
Source: [IEEE 90] in [SEI-95-MM-003].
An IEEE definition of 'Architecture'.

Related Concepts: Design *047; Design Specification *586; Design Idea *047; Architecture [Process] *499; Architecture Specification *617; Artifact *645; Systems Architecture *564.

Keyed Icon: • (delta, or a symbol for pyramid).

Architectural Description [IEEE] Concept *618

Architectural description is ''a collection of products to *document* an architecture.'' (This definition is identical with IEEE Draft Standard 1471, December 1999.)

This concept is generic and can apply to any specific architecture type.

Notes:

1. The intentionally broad term 'products' is used to include *anything*, which might be useful in describing an architecture. *Anything* can include physical models, computerized models, prototypes, blueprints, parts lists, planned test results, actual input and outputs from tests, Planguage architecture specifications, sales and training

materials, and real systems – as long as their *purpose* is to document an architecture.
2. The term 'Architecture Description' is an IEEE term, it is NOT used in the Planguage sense of a 'Description' parameter: it should really be equated to the Planguage term, 'Definition.')

Related Concepts:

- Architecture Specification *617: This concept does not include models and real systems, but only abstract specifications
- Systems Architecture *564: An architecture description can be for any specialized subset of a systems architecture, such as software or hydraulics.
- Architecture *192: this is the real set of artifacts that the architectural description describes.

Architecture Engineering Concept *499

The architecture engineering process puts in place the systems architecture, which is a controlling mechanism for the design engineering of any project.

Architecture engineering defines the strategic framework (the systems architecture), which design engineering has to work within. It lays down the standards, which help control such matters as the tradeoff processes amongst requirements. It helps synchronize design engineering disciplines across different systems.

The architecture engineering process is a *subset* of the Systems Engineering process.

Notes:

1. The architecture engineering process is distinct from the larger systems engineering process in that it is focused on *design issues*. (Systems engineering is broader. It includes consideration of the requirements, quality control, project management, and any other discipline, that is useful for satisfying requirements.)
2. The architecture engineering process is distinct from the other system level design engineering processes because it operates at a higher level, and is therefore concerned with wider issues. It has to consider the overall strategic framework and provide guidance to all the lower-level systems. It considers especially the long-term objectives, and the totality of the requirements for all systems.
3. The architecture engineering process is, ideally, technologically neutral. It should provide guidance on design, using any relevant technology, policy, motivation, organizational idea, contractual agreement, sales practice and other devices. One of the main criteria is that the architecture is cost-effective. Note that technological neutrality is not always achieved! For example, promotion of the use of standard platforms could be included within a systems architecture; and while that is an architectural decision, it is not technologically neutral.

Synonyms: Architectural Engineering *499; Architecting *499.

Related Concepts: Systems Architecture *564; Architecture *192; Requirement Engineering *614; Design Engineering *501; Architecture Specification *617.

Architecture Specification — Concept *617

An architecture specification is the written definition of an architectural component.

Notes:

1. An architecture specification either specifies a component of a systems architecture, or it specifies an architectural component of a specific system.
2. An architecture specification is a specialized form of design specification.
3. Architecture (the collective noun) is the *real* set of artifacts that the architecture specification describes. In other words, this is the *observable* architecture in a defined system. The specification may be describing *desired future* states of that system. Some parts of that specification might *never be implemented* in practice, since it serves as a vehicle to discuss architectural possibilities and options.
4. An Architecture Specification is not as broad as an Architecture Description [IEEE], which can also include models, prototypes and real systems to aid architectural description.

Synonyms: Architectural Specification *617.

Related Terms Architecture *192; Architecture Engineering *499; Systems Architecture *564; Architecture Description *618.

Assumption — Concept *002

Assumptions are unproven conditions, which if not true at some defined point in time, would threaten something, such as the validity of a specification or the achievement of our requirements.

'Assumption' is a parameter that can be used to explicitly specify any assumptions made in connection with a specific statement.

> "Assumptions are suppositions, conjectures, and beliefs which lack verification at the time of writing, or requirements and expectations that are not within our power to control, but which have been used as part of the basis for planning future actions. We identify for each the degree of risk involved and possible consequences if the assumption is erroneous."
>
> *<-Don Mills, NZ 2002 (Personal e-mail)*

Notes:

1. We need to document our assumptions systematically in order to give warning signals about any conditions that need to be evaluated, or checked, to ensure that a specification is valid. The aim is that the assumptions will be considered at the relevant future points in time, and that anyone with any additional information concerning an assumption (including lack of specification of an assumption), will volunteer it as soon as possible.
2. The purpose of the Assumption parameter is to explicitly state 'otherwise hidden' or undocumented assumptions. This permits systematic risk analysis.
3. There are many different ways in Planguage to express assumptions. Alternatives to using the Assumption parameter include using the Rationale, Condition and Basis parameters.

Example:

Hierarchic Structure [Health and Safety System]:
Type: Design Specification.
Description: A hierarchical database structure will be used.

Assumption: No negative impact on performance of Emergency and Rescue Inquiries <- JB.
Impacts: Access Response, Portability.
Is Impacted By: Available Database Packages.
Rationale: This structure is compatible with the current structure, and can be directly converted to it.
Condition: Off-the-shelf software can be used, and no in-house support is needed.
Basis: Health and Safety System required by National Law.

4. It would be good practice to specify the *consequences* of a failure for the assumption to be true. Use Impacts, Supports and similar parameters just below the Assumption statement. *See above example.*
5. It would also be good practice to specify the things that *determine* if this assumption is going to be true. Use Depends On, Authority, Source, Is Impacted By and similar parameters just below the Assumption statement. *See above example.*

Related Concepts: Basis *006; Rationale *259; Condition *024; Qualifier *124; Risk *309.

Attribute **Concept *003**

An attribute is an observable characteristic of a system. Any specific system can be described by a set of past, present and desired attributes. There are four main categories of attribute:

- Performance: 'How Good the System Is'
- Function: 'What the System Does'
- Resource: 'What the System Costs'
- Design (or Architecture): 'The Means for delivering the System'

All attributes are qualified by Conditions, which describe the time, place and events under which the attributes exist.

Attribute: "A characteristic of an item; for example, the item's color, size, or type."

Source: Dictionary of Computing Terms, *IEEE 630-90.*

Notes:

1. Performance and resource attributes are scalar (described by a scale of measure). Function and design attributes are binary (either present or absent).
2. Attributes can be complex. They can be defined by a sub-set of elementary attributes.
3. An attribute may be described by any useful set of Planguage parameters.

Example:
Reliability: "The attribute tag name."
Ambition: High duration of operation. "*Summary of the target.*"
Scale: Hours of <uninterrupted service>. "*Defining the measure.*"
Goal [Next Release]: 6,000 hours. "The required target level for the attribute."
The tag (Reliability) and the parameters (Ambition, Scale and Goal) provide a systematic framework for defining and referring to a scalar attribute's components.

Synonyms: Characteristic *003; Property *003.

Related Concepts: Performance *434; Function *069; Resource *199; Design *047.

Author **Concept *004**

An author is the person, who writes or updates a document or specification of any kind.

Notes:

1. This is a generic term, which depending on the specific document type, is usually replaced by specific roles, such as {engineer, architect, manager, technician, analyst, designer, coder, test planner, specification writer}.

Synonyms: Writer *004; Specification Writer *004.

Related Concepts: Owner *102.

Authority **Concept *005**

Authority is a specific level of power to 'decide' or 'influence' or 'enforce' a specific matter requiring some degree of judgment or evaluation. For example, the status of a specification is usually the responsibility of some 'authority' (some set of individuals holding the specific authority). Authority is often held by a specified individual or by an organizational group. A specific role may hold the authority. In addition, a document that is authorized can be used, within the document's scope, as a source of authoritative information (in lieu of access to the people holding the authority).

An Authority *parameter* is used to indicate the specific level of authority, approval, commitment, sanction, or support for a specified idea, specification or statement.

Notes:

1. This is not the same as 'Source,' which is the written or oral source of information. A Source might convey no authority whatsoever (for example, "60% <- My best guess!").

   Example:
   Past [Last Year]: 60% <- Marketing Report [February, This Year].
   Authority: Marketing Director [Tim].

Background **Concept *507**

Background information is the part of a specification, which is *useful* related information, but is not *central* (*core)* to the implementation, nor is it commentary.

Example:
In a requirement specification, the benchmarks (Past, Trend, Record) are not the actual requirements (not 'core'), but they are useful 'background' to the requirements.
The key requirement targets (Goal and Budget levels) and constraints (Fail and Survival levels) are central (core) to implementation, and are therefore not background.

Notes:

1. Parameters are clearly typed as either 'core' (for example, Scale, Meter and Goal) or 'background' (for example, Ambition, Gist and Past), or 'commentary' (for example, Source and Note).
2. Background specifications are essential to the understanding and use of a specification. However, any defects in background will *not necessarily* materially and/or negatively impact the real system. Such defects might potentially have bad impacts when used in a certain way. For example, when a Goal (core specification) is set on the basis of an incorrect Past

or Record (background specification) the resulting Goal level (a 'core' specification) will be incorrect.

Specification defects in 'background specification' are either major or minor, depending on our judgment, in the specific context, of the *potential* consequences.

Example: [Background Parameters]
• Benchmarks {Past, Record, Trend} • Owner • Version • Stakeholders • Gist • Ambition

Related Concepts: Non-Commentary *294; Core Specification *633; Commentary *632; Specification *137.

Backroom — Concept *342

Backroom is an adjective or noun, referring to a conceptual place, used to describe any processes or activities in Evo that are not necessarily visible to the Evo step recipients.

Notes:

1. Typically, Backroom is used to refer to the development and production cycles of the Evo result cycle.
2. This is where concurrent engineering takes place. Backroom activities (for example, detailed design, purchasing, construction and testing) may have to be carried out in parallel with other activities as step preparation (prior to being ready for delivery), can take arbitrary lengths of time. The overriding Evo requirement is for frequent stakeholder *delivery* cycles.
3. Evolutionary project management needs to manage the backroom and frontroom as one synchronized process.

Related Concepts: Frontroom *343.

Baseline — Concept *351

A system baseline is any set of system attribute specifications that defines the state of a given system.

Frequently, the choice of system baseline is governed by project timescales; a significant project milestone date in the past will be selected and then it is simply a case of determining the relevant individual attribute baselines on that date.

An attribute baseline is a benchmark that has been chosen for use as a start point to measure any relative system change (estimated or actual) against.

Notes:

1. Within Impact Estimation, for each scalar attribute a 'Baseline to Target Pair' is declared. The chosen baseline is usually a Past level and represents zero percentage impact (0%). In an IE table, each baseline to target pair appears immediately under the tag of its attribute on the left hand side on the table.

Example:
QX: Quality.
Scale: Time to complete a defined [Task] for a defined [Person Type].
Baseline: Past [Task = Learn to Drive Off, Person Type = Experienced Driver, Our Competitor's Product]: 1 minute.

Target: Goal [Task = Learn to Drive Off, Person Type = Experienced Driver, Our New Product]: 10 seconds.
This example shows setting a baseline and a target for a quality, QX.

Example:
ABC: IE Table [Baseline Date = Nov 7, Target Date = Dec 7].
QX:
BABC: Baseline [ABC]: Past [. . . . . . . "declare as a baseline for ABC IE table"].
TABC: Target [ABC]: Goal [. . . . . . "declare as a target for ABC IE table"].
Baseline to Target Pair [ABC]: 1 minute <-> 10 seconds. "deduced from baseline and target declarations above. Strictly not needed as repetition."
This example shows an alternative way to set a baseline and target. It introduces the idea of declaring a Baseline Date and Target Date applying across an IE table.

Table G1 ABC: IE table.

Design Idea ->	*ADI*	*BDI*	*CDI*	*DDI*
QX				
1 minute<–> 10 seconds	0%	100%	–20%	150%

Design 'ADI' has zero percentage impact, meaning that if Design 'ADI' were implemented then there would be no visible change in the quality level (it would remain at one minute and there would be no forward progress towards the target (10 seconds)).
Design 'CDI' would be even worse than the baseline and the quality level would be worse than before.
Related Concepts: Benchmark *007.

Basis — Concept *006

A basis is an underlying idea that is a foundation for a specification.
A 'Basis' parameter is used to explicitly specify a foundation idea, so that it can be understood and checked. Hopefully, if necessary, a basis specification will be challenged and corrected. It is a tool for risk analysis.
Notes:
1. Basis statements are used to declare a set of conditions, which we assume will be true. We want to make it quite clear that the related statements are *entirely contingent* upon the conditions being true.
   A Basis statement is, or will be, for the appropriate qualifier time, place and other conditions, fundamental and stable. We state a Basis in case it is untrue, or is a misunderstanding, or needs improvement in specification: the intended readership needs to check whether they agree that a Basis statement applies. In addition, we state a Basis to ensure that the conditions are checked later, at the relevant time.

2. Basis is different from Assumption. An Assumption is a set of statements, which we *expect be true* in the planning horizon (for example, the dates indicated in Goal and Fail parameters), but we *cannot be sure*; they can well change. The related specification *may* need updating if they do.
3. Basis is quite different from Rationale. A Rationale is a set of statements, which lead to a desire to make a specific specification. It explains how we got to that particular *specification*. Basis is a set of statements, which are the *foundation* on which a specification is made. If the result of evaluation of any of the relevant Basis statements changes, then the specification may no longer be valid.

Example:
Fail [A1]: 60%, [Not A1]: 50%? <- Guess as to consequence.
A1: Assumption: Drugs Law [Last Year] is still in force and unchanged with respect to this plan.
Basis: Drugs Law 'Conditions for Approval for Human Trials'<- Pharmaceutical Law [Last Year].
Rationale: Our Corporate Policy about following laws, strictly and honestly <- Corporate Ethics Policy.
Condition: Applies only to <adult, voluntary, healthy, field-trial people>.
'A1' is a defined assumption that can be reused in this or other contexts.

Synonyms: Base *006; Foundation *006.

Before — Concept *312

'Before' is a parameter used to indicate planned sequencing of events, including Evo steps and tasks.

Example:
Stage Liftoff: Step: {Ignition On Before Check Thrust OK, Ignite Motors} Before Release Tie Down.
The Evo step is planned as a sequence of step elements. Ignition On is to be done first. Followed by Check Thrust OK. Ignite Motors can be done anytime in relation to the first two, but, since it is in the brackets, it, as well as the other two events in the brackets, must be done before Release Tie Down.

Benchmark — Concept *007

A benchmark is a specified reference point, or baseline. There are two main types: scalar and binary benchmarks.

Notes:
1. A scalar benchmark is a reference level for a performance or resource attribute. It is usually used for comparison purposes in requirement specification, design and implementation.
2. A scalar benchmark is normally defined using the benchmark parameters {Past, Record, Trend}.

Example:
Usability:
Ambition: Order of magnitude better than future competitors.
Scale: Average time needed to learn to do Typical Tasks for Typical User.

Trend [Best Competitors, During New Product Lifetime, Europe Market & USA Market]: 5 minutes.
Fail [New Product, All Markets]: 2 minutes.
Goal [New Product, Initial Release]: 1 minute.
Goal [New Product, 1 Year After Initial Release]: 30 seconds.
'Trend' is the benchmark specification.

3. Function and design attributes are specified as binary benchmarks: binary attributes are either present or absent in a system.

Related Concepts: Baseline *351; Past *106; Record *127; Trend *155.

Benefit **Concept *009**

Benefit is value delivered to stakeholders.

Notes:

1. Benefits are the positive things that the stakeholders experience from a system. 'Bene' means 'good'.
2. Benefit differs from stakeholder value. Value is perceived future benefit. Value is reflected in what priority, and consequent resources, people are willing to give for something, in order to get the benefits they expect.
3. Benefit is the reality experienced in practice by defined stakeholders.
4. Benefits can include improved stakeholder environment performance, reduced costs, and improved functionality.
5. Benefits could also include the relaxation of previous constraints.
6. Systems engineering control can only be exercised over benefits, which have been specified as requirements. Reaching and keeping an unspecified benefit is unlikely!
7. Systems engineering can add value, it is up to the stakeholders to actually turn that value into benefit by exploiting the system.
8. One way to measure improvements in benefit is to extrapolate from changes in performance levels.

Synonyms: Gain *009; Profit: Informal use; Advantage: Informal use.

Related Concepts: Value *269; Performance to Cost Ratio *010; Value to Cost Ratio *635; Effectiveness *053; Stakeholder *233.

Binary **Concept *249**

Binary is an adjective used to describe objects, which are specified as observable in two states. Typically, the two states are 'present' or 'absent', or 'compiled with' or 'not complied with.'

Notes:

1. All the non-scalar attributes are binary (that is, the function and design attributes).

Related Concepts: Scalar *198.

Budget **Concept *480**

A 'budget' is a resource target: an allocation of a limited resource. A Budget *parameter* is used to specify a primary scalar resource target.

The implication of a Budget parameter specification is that there is, or will probably be, a commitment to stay within the Budget level (something which is not true of a Stretch or Wish specification).

Example:

Maintenance Effort:

Scale: Total annual Maintenance Engineering Hours per thousand lines of software code supported.
Budget [First Four Years Average]: 10 hours.
Stretch [First Four Years Average]: 8 hours.
Wish [First Four Years Average]: 2 hours.
Fail [Any Single Operational Year]: 100 hours <- Client payment limit in contract §6.7.
A Budget specification, together with 2 other resource targets and a constraint.

Notes:

1. A budget level is often arrived at through a formal budgeting process: the budget levels usually being set with regard to priorities, and available financial resources. Sometimes a budget level is determined by cost estimation, or it is determined by using competitive bidding and contracting. In some cases, the budget is absolutely fixed in advance, and we have to try to keep within it by making requirement tradeoffs or by using 'design to cost'.
2. At the very least a warning signal should be noted when a budgeted level is exceeded by a design, by an evolutionary step, or when there is a risk or threat that the budget *might* be exceeded. For example, we need to react if a resource threat to the budget level is discovered while evaluating potential alternative designs.
3. A resource target is a budget concept (small 'b' for budget). In Planguage, there are several parameters used to specify resource targets {Budget, Stretch, Wish}. The Budget parameter (capital 'B' for Budget) is used to specify the major type of resource target.

Synonyms: Budget Level *480: See Level *337; Planned Budget *480; Plan [Resource] *480: Historic usage only; Planned Level [Resource] *480: Historic usage only.

Related Concepts: Aim *001; Resource Target *436: Synonym is budget (the concept); Target *048; Stretch *404; Wish *244; Ideal *328; Goal *109.

Keyed Icon: > "A single arrowhead pointing towards the future. The same basic icon as for Goal *109, but always use an input arrow to a function oval to represent a resource attribute. In context: --->--->O

The Budget icon is the '>' on the arrow. If other levels for the resource are shown on the same arrow, the positioning of the tips of the icon symbols reflects the levels relative to each other."

Catastrophe — Concept *602

A catastrophe level of an attribute is where disaster threatens all, or part, of a system. Catastrophe can mean a variety of things such as:

- contractual non-payment performance level
- illegal quality level
- totally unacceptable level for defined stakeholders
- a level which causes the entire system to be useless (that is, worse than the survival limit).

The Catastrophe parameter can be used to specify any such known disaster level. Using the Survival parameter is another option. (These two parameters are the two sides of the same level.)

Notes:

1. The default assumption is that the catastrophe is for *the complete system.* If it is not, then qualifiers must limit its scope.
2. If a design or architecture threatens to result in any attribute being equal or worse than its Catastrophe Level, then you would discard the design, abandon or modify the requirement, or potentially abandon the project.
3. A catastrophe is not a transient failure – that is we do not expect the system to recover without some major intervention.
   Catastrophe does not imply complete irrecoverable failure. After the event, someone might change their mind and decide to 'bail out' the system. But Catastrophe, once reached, is most likely to be irrecoverable in practice.
4. A Catastrophe Range starts from the 'best' Catastrophe Level and goes in the direction of 'worse'. This can be made explicit by describing the Catastrophe Range, not just the Catastrophe Level (Describe the range by using 'or less' or 'and worse' after the numeric value).

Example:
Catastrophe [System Wide]: 60% or less.
Catastrophe [Security]: 60%.

Keyed Icon: . "A full stop. In context, a series of '.' indicates a Catastrophe Range ------->--!--------] . . . . >O . . . . [-----!-->-----------> The Survival icon (square brackets on a scalar arrow icon . . . . [----] . . . . >) can be used to emphasize the transition from Survival status to Catastrophe (non-survival) status."

Synonyms: Catastrophe Level *602; Catastrophe Limit *602; Intolerable *602; Catastrophic Failure: Informal use only; Death: Informal use only; Non-Survival: Informal use only.

Related Concepts: Survival *440; Range *552: For description of 'Catastrophe Range.'

Historical Note: The idea for Catastrophe originated from Terje Fossnes and Cecilia Haskins in August 2002.

Checking Rate — Concept *015

The checking rate is the *average* speed at which a specification is checked by a checker, using all the relevant related specifications and standards {the main specification, rules, checklists, source documents and kin documents}.

Notes:

1. The checking rate is critical for Specification Quality Control, and must normally be about 300 significant words (of checked main specification) per hour. This can vary (0.1 to 1.9 hours per 300 significant words), depending on many factors, such as the number of documents to be referenced while checking. The optimum checking rate is the checking speed that in fact works best for an individual checker to do their assigned tasks.

Related Concepts: Rate *139; Optimum Checking Rate *126.

Checklist — Concept *016

A 'checklist' for SQC usually takes the form of a list of questions. All checklist questions are derived directly and explicitly from cross-referenced specification rules. Checklists are 'stored wisdom' aimed at helping to interpret the rules and explain their application. Checklists are used to increase effectiveness at finding major defects in a specification.

Example:
STDQ: Rule: All critical project requirements must always be expressed numerically and measurably.
This is the rule. The associated checklist question is designed to help people understand how to apply the rule in practice, and identify any defects breaking the rule.
Checklist Q: Are all *performance* concepts (including all qualitative concepts – all '-ilities') expressed *quantitatively*? <- Rule.STDQ.
An example of a checklist question with the rule it supports (STDQ) being referenced.

Notes:

1. Checklists are like law court interpretations of the law. They are not the official 'law' itself, but they do help us understand the proper interpretation of the law. Anyone can write checklists at any time to give advice on how to check. They are intentionally less formal to create, and to change, than specification rules. They do not necessarily have formal 'owners.'
2. Checklists should not be used instead of a proper set of rules, which is maintained by an engineering process owner. They are only intended as a supplement for checkers. Issues can only be classified as real defects if they can be shown to violate the official agreed rules for a specification.
3. Less formal 'de facto checklists' also exist. These include any documents that can be used to check a document with a view to identification of defects. These can have other names and even other purposes than a 'pure' checklist. Examples of 'de facto checklists' include 'sources,' 'standards,' 'guidelines,' 'templates' and 'model documents.' If they help check, they must be some sort of checklist, irrespective of what people call them or intended them to be used for.

Commentary — Concept *632

Commentary specifications are remarks about other specifications. Commentary specifications will probably not have any economic, quality or effort consequences if they are incorrect: defects in commentary are almost always of minor severity.

Example:
• Note • Comment • "Text in quotes" • Source

Related Concepts: Non-Commentary *294; Core Specification *633; Background *507; Specification *137.

Complex — Concept *021

A complex component is composed of more than one elementary and/or complex component.

Notes:

1. A complex component consists of several sub-components. The sub-components may be all of the same type as the component, or of several different types.
2. Requirements, Design Ideas and Evo Steps are often complex components.

   Example:
   Goal [Alpha]: 30%, [Beta]: 20%. "This is a complex statement."
   Goal [Theta]: 50%. "This is an elementary statement."

Related Concepts: Elementary *055; Component *022.

Concept [Planguage] **Concept *188**

A Planguage concept is a formally specified idea used in Planguage.

Notes:

1. There are several types of concept found in Planguage specification. These include:
   - formal Planguage concepts defined in this glossary or other Planguage glossaries and assigned a concept number (*nnn). Some concept names are written with a Capital letter first, to signal that they are formally defined terms. Examples: Scale, Goal, and Defined As.
   - user-defined terms.
2. A Planguage concept, once defined, can be referenced by any useful synonyms or identifiers. These include tags, keyed icons, drawn icons, abbreviations, synonyms, acronyms and alternative language terms (for example, German or Japanese terms).
3. The central idea of a Planguage concept is that the concept itself is independent of the particular means (pointer, reference, cross-reference, tag, icon, concept number) that we choose to apply in order to reference that concept. We can focus on the concept, and not the particular term, about which people might disagree or have cultural difficulties in accepting.
4. Defined concepts can be:
   - reused without explaining them again
   - redefined by Planguage users locally (which simultaneously changes (hopefully improves) the definition of all the other terms, which reference the defined concept)
   - referenced by a set of terms in any language, without necessarily having to rewrite the concept definitions themselves in that language. For example, the concepts could be defined in English, but a Norwegian set of pointers to the concepts can be quickly defined, to permit teaching or multinational project learning and use of specifications.

Example:
Begrep [Norwegian Bokmål] = *188 "Concept [Planguage, US]."
Tilstand [Norwegian Bokmål] = *024 "Condition [Planguage, US]."
Marked [Norwegian Bokmål] = Market [Corporate Glossary].
Is [Norwegian Bokmål] = Ice Cream [Project XYZ Glossary].
*In these examples Planguage concepts like *188 are given a foreign language name ('Begrep') in Norwegian. Using the synonym, 'Begrep,' a user can access the full definition in English.*

Synonyms: Planguage Concept *188.
Related Concepts: User-Defined Term *530; Planguage *030.

Condition **Concept *024**

A condition is a specified pre-requisite for making a specification or a system component valid.

Notes:

1. Evaluation of the status of a condition can be carried out anytime, and on many different occasions, each with a potentially different result.

The result of an evaluation of a condition is the 'current condition status' or more simply, 'status.'

2. *Evaluation* of a condition will determine if its status is currently true or false.
3. There are several distinct kinds of conditions:
   - Reusable Conditions
   - Qualifier Conditions
   - Pre-requisite Conditions

   Reusable Conditions:

   The Planguage *parameter* 'Condition' is used to define *conditional terms*. The 'true or false' status of such a term can be determined when required. This parameter statement can be used to define:

   - reusable conditions (conditions that many other statements can make use of)
   - conditions which are complex, and get simplified by having a single tag to express them.

   Example:

   Senior: Condition {Senior Citizen Or Service over 20 years to Company}.

   Pass Through: Condition: Traffic Light {Green, Yellow, Blinking Yellow, Not Red}.

   Qualifier Conditions:

   One or more qualifier conditions can be used to specify a statement *qualifier* (for example, '[End of March, USA, If Peace]'). A statement qualifier must be completely true for the qualified statement to be valid.

   Example:

   Level X: Goal [A, B, C]: 33%.

   Note: Level X is only a valid Goal when all three qualifiers {A & B & C} are true/valid.

   Another example, a Goal level specification is only valid when all the conditions in its qualifier are true. The qualifier in the Goal statement below has three conditions.

   Example:

   Goal [Year = Release + 1 Year, Market = Europe, Not War]: 66%.

   A qualifier condition may consist of an explicit tag name with an appropriate variable declared (for example, 'Market = Europe.' 'Market' is the tag name and 'Europe' is the variable).

   If there is no ambiguity, the tag name may be implied and simply the variable is stated.

   A qualifying condition may, or may not, be satisfying a Scale qualifier.

   Example:

   Learning Time:

   Scale: Time in minutes for a defined [Role] to <learn> a defined [Task].

   Goal [Task = Login, Role = Operator, Country = Spain]: 2 minutes.

   In the above example, '[Task = Login, Role = Operator]' is a statement qualifier.

   Both 'Task' and 'Role' are qualifier conditions. They are also both Scale qualifiers. Task is assigned a variable of 'Login,' and 'Role' a variable of 'Operator'.

   'Country = Spain' is an additional qualifier condition, which has been added. It is not a Scale qualifier.

If the Task under consideration is 'Login,' the Role is 'Operator' and the Country is Spain, then the target goal for consideration must be 2 minutes. In other words, the evaluation of the statement qualifier as 'true' depends on all its qualifying conditions being 'true.' Each qualifier condition is only true if its variable matches the specific instance being considered (Task is 'Login,' Role is 'Operator' and Country is 'Spain'). Each qualifier condition might have a set of valid variable settings. For example, Country: {Spain, USA, Germany}.

Pre-requisite Conditions:
A set of conditions, can be used as a prerequisite for a system component, such as entry to a defined process, or exit from a defined process, or use of a product. Any such conditions should be explicitly listed as pre-requisites or qualifications.

Example:
Exit Conditions:
X1: Senior. "See definition in above example."
X2: Level X. "Not only A & B & C, but also 33% Goal reached."

Example:
Process: Evening Closedown [Application: Default: ABC].
"The square brackets, '[ ]', specify a qualifier condition. It asks the question: Which application is this generic process being applied to?"
Gist: Application process for evening closedown for the night.
Entry Conditions:
E1: All users have logged off. "A condition. Are all users logged off: true or false?"
E2: After 8pm. "Another condition. Is time after 8pm: true or false?"
Procedure
... "If all entry conditions are met (that is, are 'true'), then it is 'valid' to carry out the process."

Synonyms: Conditional Term *024; Pre-Requisite Condition *024.
Related Concepts: Condition Constraint *498; Qualifier *124; Status *174: The result of the evaluation of a condition.
Keyed Icon: [<*condition tag name*>]

Condition Constraint — Concept *498

A condition constraint is a requirement that *imposes a conscious restriction* for a specified system scope. A condition constraint, also called a 'restriction,' is a binary type of *requirement*.

Notes:

1. A condition constraint differs from a 'condition' in that some kind of failure, invalidity, problem, dependency, risk, or other problem may be experienced, if the constraint is not met. It serves as a warning signal for problems.

   Example:
   CCR: Constraint [Release 1]: Initial product must be delivered before the end of January.
   Rationale: Financial penalties apply if this contractual deadline is not met <- Contract Section 2.4.

2. A condition constraint can be categorized by innumerable useful categories, but some common ones are design, legal, cultural, market,

geographic, safety, and language. (Note these categories can also apply to other types of constraint, for example, a certain level of reliability – a scalar performance constraint – could also be a 'legal constraint').

Example:
C1: Constraint [Language]: All official languages of a market will be fully supported in the user interface, and all training and handbook information.
C2: Constraint [Safety]: All Electrical Equipment brought onboard any Corporate Aircraft as Standard Kit will comply with Corporate Electrical Safety Standard 1.5.

Synonyms: Restriction *498.
Related Concepts: Condition *024; Constraint *218; Status *174: Synonym is 'State'.

Consists of — Concept *616

'Consists Of' is a parameter used to list a *complete* set of the sub-components or elements comprising a component.

Example:
Security:
Consists Of: {Integrity, Attack}.
Alternatively, Security = {Integrity, Attack}.

Related Concepts: Includes *391 "Used to list *some*, but not necessarily all components"; Element *022.
Keyed Icon: = {...} "Is equal to the set."

Example:
Core Family = {Mother, Father, Children}.

Constraint — Concept *218

A constraint is a requirement that *explicitly* and *intentionally* tries to directly restrict any system or process. A key property of a constraint is that a penalty or loss of some kind applies if the constraint is not respected.
Constraints include limitations on the engineering process, a system's operation, or its lifecycle.

"A constraint is 'something that restricts'."
(The American Heritage Dictionary (Dell))

Notes:

1. There are two kinds of constraints: Binary and Scalar.
   Binary constraints are statements that tend to include the words 'must' or 'must not': that is, they tend to make demands about what is mandatory for the system or what is prohibited for the system. Binary constraints are declared either by using a Constraint parameter or by specifying 'Type: Constraint.'

   Example:
   C1: Constraint: A design idea must not be made of material, components or products only produced outside the European Market, if there is any EU material which can be used.
   C2: Constraint: The design must contain ideas based on our own patents whenever possible.

C3:
Type: Constraint.
Defined As: All stakeholder critical qualities must be planned and delivered so that they are viewed as obviously and significantly better than any competitor in the same price range.

From an attribute viewpoint, binary constraints are function constraints, design constraints or condition constraints.

Scalar constraints are specified for performance or resource attributes. They are specified using Fail and Survival parameters, which set the constraint levels on a scale of measure.

2. All engineering specifications (requirements and design) and management plans, once stated, potentially and probably have some constraining influence on the rest of the planning or engineering process. So all specifications and plans are 'constraints' in this sense.

   However, we can *clearly distinguish* between specifications where the *primary* intent is to constrain (like a mandatory constraint or a level for Survival), and those specifications where the primary idea is *not to constrain*, but to *motivate positively* (for example, a binary function target or, a scalar performance target, such as a Goal or Stretch level). We could classify the former as 'intentional and direct constraints' and the latter as 'unintentional and indirect constraints'.

   So, we only classify specifications and concepts as 'constraints' when the clear intent and primary purpose is to restrain, limit, restrict constrain or stop.
3. You have to look at constraints from a 'stakeholder' viewpoint. As with any requirement or design, what is a requirement for one level of system stakeholder, is a constraint as viewed by sub-ordinate stakeholder levels.

   One stakeholder's requirement is another stakeholder's constraint.
4. All constraints are valid for their associated defined conditions. Sometimes the constraint conditions are stated explicitly, sometimes they are implied or inherited from more global specifications.

   A 'global' constraint is usually imposed by higher levels of authority, or by earlier planning processes. For example: by company policy, law, contract, strategic planning or systems architecture.

   Example:
   An example of a global constraint:
   Availability Criteria: Constraint: No Company Product shall ever be designed with less than 99.5% availability.
   A corresponding local constraint setting a higher constraint level:
   Fail [US Market, Military Systems]: 99.98%.
5. Constraints can be classified by the 'relative level of organization' they apply to, as proposed by Ralph Keeney (1992):
   - Fundamental Constraints: handed down to us from higher authority
   - Strategic Constraints: ones we have imposed at our own level, over which we have control
   - Means Constraints: constraints imposed at levels supporting us, which we can therefore overrule.
6. All requirements, including all constraints, have different 'priority.' This priority (or 'power') is determined by the conditions (the

qualifiers) and by their related specifications (for example, by parameters like ‘Authority’). It is a complex process to determine constraint priority, and the ‘answer’ is dynamically changing. There is probably no absolute constraint that must be respected ‘no matter what.’ That is, there might always be a higher priority consideration that overrides a given constraint. For example, ‘Thou shalt not kill (except in self-defense).’

7. Constraints always represent, in some way, some of the values of some stakeholders. But a given constraint does not necessarily agree with the values of all stakeholders. The constraint of one stakeholder might be in direct conflict with the requirements of another stakeholder.
8. Any set of categories (including no categories!) can be specified for classifying constraints. System, performance, budget and design are an arbitrary few such categories.
9. I view constraints as borders around a problem. We can do anything we like within the borders, but we must not wander outside them.

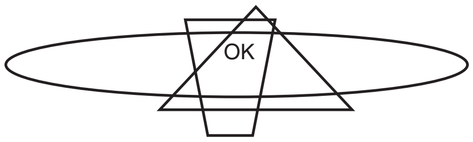


Figure G3
Constraints impose restrictions on both other requirements and designs. But the remaining space (‘OK’) gives considerable freedom to set more-exact requirements, and to specify more-exact designs.

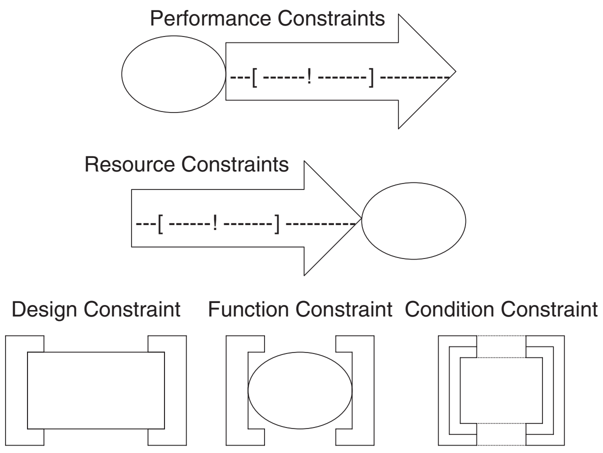


Figure G4
Drawn Icons for Constraints.

10. A requirement is a 'constraint on succeeding engineering processes' if and only if it has an authority, or other form of priority, which means that:
    - you must stay within its guidelines (when making later decisions or specifications)
    - you cannot change it yourself (in order to avoid obeying it), without authority to do so.

Related Concepts: Requirement *026; Function Constraint *469; Performance Constraint *438; Resource Constraint *478; Design Constraint *181; Condition Constraint *498; Survival *440; Fail *098.
Keyed Icons: For Survival: [and/or] and for Fail: !
"In context: ----[----!---]---->O---[-------!---]----->"

Continuous Process Improvement **Concept *424**

Continuous Process Improvement (CPI) includes any and all continuous long-term effort to systematically improve an organization's work processes.
Acronym: CPI *424.
Related Concepts: Defect Prevention Process (DPP) *042; Statistical Process Control (SPC) *466.

Core Specification **Concept *633**

Anything classed as, 'core specification,' will result in real system changes being made: incorrect core specification would materially and negatively affect the system in terms of costs, effort or quality. Specification defects in core specification are almost always of major severity.
Example:
Core Specification Parameters include:
• Scale • Meter • Goal • Definition • Constraint
Notes:
1. Core specification can be distinguished from 'commentary' and 'background' (supporting) specification.
2. Core specification is the 'meat' in specifications of requirements, designs, Evo steps and test cases.

Synonyms: Implementable Specification *633.
Related Concepts: Non-Commentary *294; Background *507; Commentary *632; Specification *137; Specification Quality Control (SQC) *051.

Cost **Concept *033**

Cost is an *expense* incurred in building or maintaining a system. It is consumption of a resource.
Synonyms: Price *033; Expense *033: The degree of consumption, how much resource was used.
Related Concepts: Resource *199.
Keyed Icon: -|->O
"The keyed icon is a level symbol, '|', set on a resource scale of measure, '-->O'."
Note: The neutral symbol '|' is chosen to represent the generic cost concept, rather a currency symbol, because the resources involved are more than just money. If you want to link the icon to the idea of a cost range, then think of the chosen symbol as a minus sign, turned 90 degrees.

Example:
---|||||||--->O expresses a cost range.
The keyed icon for Cost [Money] is a currency symbol, default € (Euro).

Example:
--€-->O to express a money cost and ----<€€€€€€€€€|======>O to express a cost range.

Credibility **Concept *035**

Credibility expresses the strength of belief in and hence validity of, information. Within Impact Estimation, credibility is usually assessed for the evidence and sources supporting each specific impact estimate. Credibility is expressed as a numeric value on a range of credibility ratings from 1.0 (for perfect credibility) to 0.0 (for no credibility at all). These credibility values can be used to credibility-correct the impact estimates: each impact estimate is multiplied by its relevant credibility.

Example:
If an impact estimate were 40% and its credibility were 0.5, then the credibility-adjusted estimate would be 20% (40% multiplied by 0.5).

Critical Factor **Concept *036**

A critical factor is a scalar attribute level, a binary attribute or condition in a system, which can on its own, determine the success or failure of the system under specified conditions.

Notes:
1. A critical *failure* factor is usually specified as a constraint level (for example a 'Fail' or 'Survival' level), or as a binary constraint ('Constraint').
2. A critical *success* factor is usually specified as a target level (a 'Goal' or 'Budget' level), or as a binary target ('Target').

Related Concepts: Critical Success Factor *418; Critical Failure Factor *025.

DDP **Concept *041**

Acronym for Defect Detection Process *041

Defect Detection Process **Concept *041**

The Defect Detection Process (DDP) is part of Specification Quality Control (SQC), which also includes the Defect Prevention Process (DPP). It is the systematic, project-focused process of identifying specification defects.

Source: A detailed description of the DDP process can be found in (Gilb and Graham 1993) and (Wheeler, Brykcznski and Meeson 1996).

Rationale: This is to avoid the high cost of late defect removal (at test, or in field), or to avoid the high cost of the consequences of malfunctions caused by the defect: "A stitch in time saves nine."

Notes:
1. The DDP is 'project oriented' in that it is primarily concerned with a project's economics, rather than an organization's work process economics (that is the DPP concern).
2. DDP is not itself concerned with process improvement, but it provides a stream of data, concrete defect examples, and a working environment that can be used to feed into, and to help analyze effects of, a Defect Prevention Process.

Acronym: DDP *041.
Related Concepts: Specification Quality Control (SQC) *051; Defect Prevention Process (DPP) *042.

Defect Prevention Process **Concept *042**

The Defect Prevention Process (DPP) is a specific IBM-originated process of continuous process improvement. It is part of Specification Quality Control (SQC).

Notes:

1. The DPP process works towards continuous process improvement for ongoing, and especially future, projects in a larger organization. It is fed suggestions, and data, on problems, from the Defect Detection Process (DDP), and from other defect-identification sources, like testing and customer feedback.

   "An ounce of prevention is worth a pound of cure."

2. As reported by IBM, in organizations of 100 to 1,000 people, about 200 to 1,000 process changes may be implemented annually. On initial DPP implementation (the first project), 50% of the total number of historical defects may be eliminated in the first year of use and 70% eliminated within 2–3 years.

Sources:

- Inspired by classical Statistical Process Control ideas (Deming 1986), the Defect Prevention Process (DPP) was developed and refined (from 1983 onwards) by Carole Jones and Robert Mays of IBM Research Triangle Park NC with the aim of improving IBM's processes for software engineering, hardware engineering and administration (Mays 1995).
- A detailed description of DPP can be found in (Gilb and Graham 1993 Chapters 7 and 17).
- DPP was the direct inspiration for IBM assessment process Level 5 (Ron Radice cited in Mays 1995), US DoD Software Engineering Institute's Capability Maturity Model, CMM Level 5, and CMMI Level 5.

Acronym: DPP *042.
Related Concepts: Plan-Do-Study-Act Cycle (PDSA) *168; Specification Quality Control (SQC) *051; Defect Detection Process (DDP) *041; Continuous Process Improvement *424; Process Improvement *114; Process Improvement Suggestion *088; Process Meeting *119; Process Change Management Team *118.

Definition **Concept *044**

'Definition' or 'Defined As' is a parameter that is used to define a tagged term.

Notes:

1. The tagged term could then be re-used anywhere else within scope, and would always have this precise definition.
2. Any tagged statement is, in practice, a definition of that tag, so the use of the 'Defined' parameter is to make it *explicit* that the statement is intended as a reusable definition.

   Example:
   Trained: Defined As: At least 30 hours classroom, and 'passed' practical exam.
   Trained: At least 30 hours classroom, and 'passed' practical examination.

Trained: Def: At least 30 hours classroom, and 'passed' practical examination.
Trained: Definition: At least 30 hours classroom, and 'passed' practical examination.
Trained = At least 30 hours classroom, and 'passed' practical examination.
Equivalent definitions of the term, 'Trained.'

Example:
Reliability:
Scale: Hours to <complete> defined [Task: Default = Most Complex Task].
Fail [USA]: 5 hours. "Most Complex Task is suitable default for a Fail specification."
Goal [Europe, End Next Year]: 10 hours.
Most Complex Task: Defined As: The work task, which normally takes most employees longest clock time to complete on average.

Abbreviations: Def *044.
Synonyms: Defined *044; Defined As *044.
Keyed Icons: : *or* = "Whatever follows the icon symbol is a definition of the tag or parameter to the left of the symbol. '=' is less commonly used."

Dependency — Concept *189

A 'dependency' is a reliance of some kind, of one set of components on another set of components.

Notes:

1. Any given component can be part of numerous dependencies (either having one or more dependencies, and/or having one or more dependencies placed on it).
2. The reliance involved in a dependency can be of many kinds. For example, there can be dependency for operation, for success, or for failure avoidance.
3. Qualifiers specify *implied* dependencies.
4. The parameter, 'Dependency,' or its synonym, 'Depends On,' is used to *explicitly* specify a dependency.

Example:
Z [T]: YY. "Only if T is true, does Z have a value of YY."
A: Depends On: B. "If B is not true then A is not true."
Tag A:
Dependency: XX. "Tag A has a dependency on XX."

Example:
Goal [Contract Beta]: 60%.
This means that the Goal level requirement of '60%' is valid as a Goal if, and only if, 'Contract Beta' is 'in force.' The Goal has an implied dependency *on the qualifier, 'Contract Beta.'*

Example:
Dependency: The satellite must be operational for the phone to operate. <- Catherine.

Example:
Dependency XX: Design Idea XX -> Reliability [USA, If Patent PP].
Example of dependency of an objective (Reliability [USA]), on both a design idea (Design Idea XX) and a condition (Patent PP). Note: -> = 'Impacts'.

This is a 'weak' dependency statement because we have no specification of whether the dependency is trivial or critical in degree. A numeric impact estimate could be used to give us that information, later, if we wanted it.

Example:
Tag: Refugee Transport.
Type: Function.
Description: Moving refugees back to home villages.
Source: Charity Aid Manual [March, Last Year].
Depends On: The mode of transport will be determined by safety and cost factors.
Or the equivalent:
Dependency: The mode of transport will be determined by safety and cost factors.

Example:
Contract Beta: Depends On: Conglomerate Corp [Our Customer, USA].
This means that if 'Conglomerate Corp [Our Customer, USA]' is not true, then 'Contract Beta' is not 'true.'

Rationale: To promote awareness of relationships, ensure more realistic planning, and provide the ability to identify reliance, and therefore cope with any associated risks.
Synonyms: Depends On *189.
Related Concepts: Before *312; After *313; Impacts *334; Is Impacted By *412.

Description — Concept *416

A description is a set of words and/or diagrams, which describe, and *partially* define, a component.
The parameter 'Description' is used to specify description.
Notes:
1. A description will convey the essence of a concept, or of a specification, but *the full definition* of the element may well require many other parameters to define it fully (from all interesting viewpoints), including implied or inherited definitions from other system components.
2. Models, real systems and prototypes can also provide a form of 'description.'

Example:
Mechanical Power:
Type: Function.
Assumption: At least 100 horsepower.
Constraint: Product Cost is lower than €500.
Risk: Last of European Commission Development Funding.
Version: March 1, This Year.
Description: The mechanical component that will provide all mechanical power to the system.
Note that the function description is only one part of the full function definition of 'Mechanical Power.'

Design Constraint — Concept *181

A design constraint is an explicit and direct restriction regarding the choice of design ideas. It either declares a design idea to be compulsory (Mandatory Design) or to be excluded (Prohibited Design).

Design constraints are dictated from an earlier system development stage (a higher level or a more specialized level). For example, the system architects pass on a number of design constraints, within the architecture specifications, to the system engineers.

A design constraint is a binary requirement. It can be a generic constraint or involve specific design(s).

Example:

=================== Prohibited Designs ===================

P1: Constraint: Products and Services of direct competitors shall be avoided.

P2: Constraint: No software product version shall be released for sale until at least 3 month field trial has completed reporting no major faults outstanding <- Technical Director's Policy 6.9.

P3: Constraint [Europe]: No goods will be shipped without advance payment or bank guarantee.

================== Mandatory Designs ==================

M1: Constraint: Resident Workers in Country of Export shall be used wherever possible.

M2: Constraint [IT Projects, In House]: Commercial Off The Shelf Software shall be used exclusively.

M3: Constraint: Products and Services from Our Corporation, Our Customers and Partners are preferred <- Corporate Policy 5.4.

M4: Constraint [Programming]: Use Java as Programming Language.

Notes:

1. Some people use the term 'Design Constraint' to mean anything that constrains the choice of design. However, within Planguage the term is more restricted. It is a direct constraint on design ideas themselves; directly referring to design ideas, generically or specifically. All other types of requirements 'constrain' our choice of design, but not as directly as a design constraint.

   Indirect Constraints:

   - A Resource Constraint determines resource, and so impacts optional design.
   - A Performance Constraint determines performance, and so impacts optional design.
   - A Function Constraint determines function, and so impacts optional design.
   - A Condition Constraint determines conditions, and so impacts optional design.

   Direct Constraints:

   - Design Constraints determine design directly, by specifying a mandatory design or a prohibited design.

     All requirement types: targets and constraints – have some potential effect on our design choices. But, design constraints are 'direct' in the sense that they make specific design decisions.

     Example:

     Spruce Goose:

     Type: Generic Design Constraint.

     Definition [If Wartime]: A troop transport plane may not use scarce <metal alloys>.

Howard Hughes' airplane, 'The Spruce Goose,' had this design constraint before the end of the Second World War. He made the plane largely of 'spruce' wood.

2. Only designs that are 'design constraints' should be allowed within *requirement* specifications. All *optional* design ideas, designs you can swap out if you find a better one, should be specified in *design* specifications. This is so that each level of design responsibility knows what it is free to do, and not free to do.

Synonyms: Architectural Constraint *181; Design Restriction *181; Constrained Design: Informal Use; Required Design: Informal Use; Solution Constraint: Informal Use.

Related Concepts: Constraint *218; Requirement *026; Design Idea *047; Condition Constraint *498.

Design Engineering — Concept *501

Design Engineering is an iterative process of determining a set of designs, with rigorous attention to quantified and measurable control of their impact on requirements.

The design engineering process implies the matching of potential and specified design ideas with quantified performance requirements, quantified resource requirements, and defined design and condition constraints.

Notes:

1. Planguage involves design engineering. By contrast, conventional 'design' activity (the kind of 'design' often found in the literature and in practice) usually has a less systematic, less quantified process, using perhaps intuition, tradition, and more trial and error, to determine satisfactory technology and to determine stakeholder satisfaction. It is characterized by naming objectives (for example, 'better usability'), and naming designs (for example, 'single standard interface'), but not following up with *quantified* versions (that is, providing the information captured in Planguage, using such parameters as Scale, Goal, Impact Estimate and Meter).

Related Concepts: Design Process *046; Architecture Engineering *499; Systems Engineering *223; Engineering *224.

Design Idea — Concept *047

A design idea is anything that will satisfy some requirements. A *set* of design ideas is usually needed to solve a 'design problem.'

Notes:

1. A design idea is not usually a requirement. However, a design idea can be a requirement if it is a *design constraint.* That is, a specific design is stated as mandatory or prohibited in the requirements.
2. Requirements are inputs into a design process; design ideas are the outputs.
3. A design idea can, in principle, be changed at any time for a 'better' design idea (without having to ask the permission of any stakeholders because the system designers are responsible for the proposed design ideas). A 'better' design meets the requirements by giving more performance and/or less cost.
4. A satisfactory design idea can have some negative performance scalar impacts, and still be acceptable overall. As long as the negative impacts (negative side effects) of a design idea do not prevent us from reaching all the required target levels, the design idea can be used.

Figure G5
The drawn icon for a Design Idea *047.

5. A design specification is a written definition of a specific design idea. (See also the design specification template.)

Synonyms: Design *047; Strategy *047; Proposed Solution *047; Means *047.
Related Concepts: Architecture *192; Policy *111; Design Constraint *181; Design Specification *586; Design Problem *048.
Drawn Icon: A lying-down rectangle. (The standing rectangle is a document icon.)

Design Process — Concept *046

The design process is the act of searching for, specifying, evaluating and selecting design ideas, in an attempt to satisfy specified stakeholder requirements.
Design is finding a set of solutions (design ideas) for a set of defined requirements.
Overview of the Design Process:

- Analyze the Requirements
- Find and Specify Design Ideas
- Evaluate the Design Ideas
- Select Design Ideas and Produce Evo Plan

Design can be carried out in several ways. It can be based on tradition, on intuition, on dogma, on principles or heuristics. It can also be based on multidimensional quantified logic – this latter we would call 'engineering' or 'systems engineering.'

> "Design comes about entirely from the playing out of the evolutionary algorithm." <-*Susan Blackmore.*[1]

Related Concepts: Design Engineering *501; Systems Engineering *223; Engineering *224.

Design Specification — Concept *586

A design specification is the written specification of a design idea. A set of design specifications is the main output of a design engineering process. A specific set of design specifications, when implemented, will, to some degree, meet the stated requirements.
Notes:

1. A set of design specifications attempts to solve a design problem. Identification and documentation of the individual design ideas, and their potential contribution towards meeting the requirements, helps selection of the 'best' design ideas for implementation.

[1] Blackmore, Susan, *The Meme Machine*, Oxford: Oxford Paperbacks, 2000, ISBN: 019286212X. See Page 205.

2. The design engineering process uses the requirement specification as input. The design engineering process output is a set of design (solution) specifications (of design ideas).
3. See the design specification template for details of the required specification data.
4. The design specifications might contain information about the *expected* attributes of the designs for meeting requirements. This 'expected attributes' information of a design specification might be in the form of an Impact Estimation table or, it can be as simple as an assertion of impacts on requirements, referenced by their tags (see example below).

Example:
Engineer Motivation:
Gist: Motivate, using free time off.
Type: Design Idea.
Impacts [Objectives]: {Engineering Productivity, Engineering Costs}.
Impacts [Costs]: {Staff Costs, Available Engineering Hours}.
Definition: Offer all Engineers up to 20% of their Normal Working Hours per year as discretionary time off to invest in Health, Family and Knowledge {Studies, Write Papers, Go to Conferences}.
Source: Productivity Committee Report 1.4.3.
Implementor: Human Resources Director.

Template: Design Specification Template.
Abbreviations: Design Spec *586.
Synonyms: Technical Design: Informal use; 'The Design': Informal use.
Related Concepts: Design Engineering *501; Design Idea *047; Systems Architecture *564; Architecture *192; Architecture Specification *617; Specification *137.

Deviation **Concept *475**

Deviation is the amount (estimated or actual) by which some attribute differs from some specific benchmark or target. Deviation is usually expressed numerically using either absolute or percentage difference.
Synonyms: Variance *475.
Keyed Icon: ±

DPP **Concept *042**

Acronym for Defect Prevention Process *042

Due **Concept *554**

'Due' is a parameter indicating when some aspect of a specification is due.

Example:
Due [Sample A]: End of January Next Year <- Contract Section 3.5.6 [Supplier X].

Synonyms: Deadline *554; Due Date *554.
Historical Note: The idea of 'Due' as a parameter was from an unpublished note by Jens Weber, Daimler Chrysler, Frankfurt.

During **Concept *314**

'During' is used when specifying events (including Evo steps and tasks) to indicate a time dependency for events that must be carried out concurrently (that is, done in parallel).

Example:
Step 33: Step: {A During B During C}. "Do A, B and C concurrently."

Elementary **Concept *055**

An 'elementary' component is not decomposed into sub-components.

Notes:

1. A component can be elementary because it is unable to be decomposed into sub-components, or because there is no declared intent to decompose it.
2. The decision to subdivide a complex concept into elementary concepts is a practical and economic matter. It depends on:
   - the size and complexity of a project
   - the need for precise control over system attributes
   - the risks taken if specification detail is inadequate
   - engineering culture
   - intellectual ability to decompose
   - other factors.

   Even when an initial decision is made about having no further decomposition of an idea, later events and opportunities, or later more detailed phases of systems engineering, may cause a concept to be decomposed into elementary concepts.

   The reverse can be true too. Initial decomposition may seem unnecessarily detailed or unnecessarily constraining. So, the concept may be simplified from a complex concept back to an elementary concept.
3. The essential characteristic of scalar attributes, which tells us if they are elementary, is the number of defined scales of measure. There is *only one* distinct Scale for each elementary concept.
4. Elementary concepts are directly measurable or testable. You can only test or measure a complex concept by way of testing and measuring the set of its elementary concepts. A complex concept is not the 'sum,' but the 'set' of its elementary concepts.
5. Normally an elementary statement can have its own distinct 'tag,' and can be treated (developed, tested, costed, quality controlled) relatively independently of any other elementary statement.

Related Concepts: Complex *021.

Error **Concept *274**

An 'error' is something done incorrectly by a human being.

Notes:

1. Errors are usually committed unintentionally; they are often forced to happen by 'bad' work processes (Statistical Process Control Theory (Deming 1986; Juran 1964; 1974)).
2. Human errors in specification processes lead to defects in specification or evaluations. For example, errors in systems engineering processes result in (written) engineering and contractual specification defects. In turn, specification defects result in faults in the system, which may or may not, result in system malfunctions (the fault actually occurs). See related concepts.

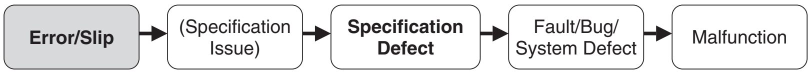


Figure G6

3. SQC is a means of checking specifications to discover whether errors have been made. Any suspected violation of any applicable specification rule is logged as an issue.

“ To err is human.”
Saying, and similar to Plutarch (AD 46–120) “ For to err in opinion, though it be not part of wise men, is at least human”.

Synonyms: Slip *274.
Related Concepts: Specification Issue *529; Specification Defect *043; Fault *339; Malfunction *275.

Estimate **Concept *058**

An estimate is a numeric judgment about a future, present or past level of a scalar system attribute. (This includes all performance and cost attributes.)
Notes:
1. Estimates are usually made where direct measurement is:
   - impossible (future), or
   - impractical (past), or
   - uneconomic (current levels).
2. An estimate is usually extrapolated from available information, and past experience.
3. An estimate can be made for numeric facts from the *past* (benchmarks), even if precise past data is not available.
4. Estimates are made about any scalar system attribute: cost levels, resource availability, quality levels, savings and other dimensions of systems.

Estimate, To **Concept *059**

In Planguage, to ‘estimate’ is:
The process of arriving at a judgment by guessing the probable numeric value of a numeric attribute level using other methods than immediate measurement.

Estimate: “to judge or determine generally but carefully (size, value, cost, requirements, etc.); calculate approximately.”
Webster’s New World Dictionary

Estimation is not to be confused with Quantification or Measurement. (See figure in concept, ‘Quantify, To *385’.)
Related Concepts: Estimate *058; Quantify, To *385; Specify, To *239; Measure, To *386.

Event **Concept *062**

An event is a specified occurrence.
Example:
- President Inaugurated
- Process Begun
- Process Ended
- Task Started
- Task Interrupted
- Contract Signed

Notes:
1. ‘Event’ is not used here in the ‘organized occasion’ sense of the term. In other words, it is not used in the sense of a ‘wedding’ or a ‘gala opening of a building’ being an ‘event.’

2. An event is not the ‘carrying out’ of an activity, such as performing a task, a process, or some systems engineering implementation (like implementing an Evo step).
3. An event must be clearly distinguishable from the non-occurrence of the event. It must be reliably observable, testable or measurable in the real world. Consequently, events must be precisely defined – unambiguously.
   However, the occurrence of an event is not the same as our measurement of it. An event occurs whether or not it is immediately detected or measured.
4. An event usually results in some measurable change in a status. If a specific event has occurred, then any status associated with the event will have changed. By evaluating the relevant event condition, the setting of a status can be determined.
5. An ‘event condition’ can be defined as a qualifier condition – the event must have happened for the qualifier condition to be true.
   Example:
   Goal [First Sale]: 9% Or Better.
   First Sale: Event: We make our first sale of refrigerators to the USA.
   Past [Last Year, Europe, First Flight]: 98%.
   First Flight: Type: Event. Description: Successful first flight <officially logged>.
6. To attempt a more detailed definition:[2] An event is an occurrence (a set of circumstances, which include changes in status), localized in space and time, which results from some activity and which is significant as an indicator of progress or as a stimulus (acts as a trigger) for other activity.
7. An event can be defined in time and space (theory of relativity), but it can be conceived of, specified and defined without time and space coordinates.

Synonyms: Happening; Occurrence; Point (in space-time).
Related Concepts: Qualifier *124; Condition *024; Status *174.

Evidence — Concept *063

Evidence is the historic facts, which support an assertion. The evidence usually will have been the basis for making an assertion.
In Impact Estimation, evidence is required for each impact estimate. Where there is no evidence, it should be clearly stated that there is none.

Example:
Design B -> Goal 1.
Scale Impact: 10 minutes.
Evidence: Of 100 surveyed Customers last year, 30 agreed there was this level of impact on Goal 1 <- Marketing Report A123.

Evo — Concept *355

Abbreviation for Evolutionary Project Management *355 and Evolutionary *196.
Readers will have to bear with me that I use this abbreviation for both ‘Evolutionary Project Management’ and ‘Evolutionary.’ The underlying concept is the same <-TG.

[2] With thanks to Don Mills, New Zealand.

Evo Plan **Concept *322**

An Evo plan is a set of sequenced and/or a set of yet-to-be sequenced Evo steps. The current planned sequence of delivery of any of the steps should be reconsidered after each Evo step has actually been delivered and the feedback has been analyzed. Many factors, internal and external, can cause a re-sequencing of steps and/or the insertion of previously unplanned additional step(s) and/or the deletion of some step(s).

It is the identification of the *next* Evo step for delivery that should be our focus for detailed practical planning. After all, at an extreme, the other planned steps may never be implemented in practice.

Notes:

1. For the Evo steps, an Evo plan is likely to specify only the tag names. Detailed specification of an Evo step will be held in its step specification.
2. An Impact Estimation table may be included in an Evo plan. Such a table would hold the estimates for the impacts of each of the Evo steps, and the feedback after delivery of each of the Evo steps. The progress made could then be tracked against the Evo plan estimates.

Synonyms: Evolutionary Plan *322; Evolutionary Delivery Plan *322.

Related Concepts: Evolutionary Project Management (Evo) *355; Evolutionary *196.

Drawn Icon: A series of any number of steps, each one representing an Evo step.

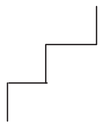


Figure G7
The drawn icon for Evo Plan *322 consisting of several Evo Steps *141.

Evo Step **Concept *141**

An Evo step ('evolutionary step' or simply 'step') is a 'package of change,' containing a set of design ideas that on delivery to a system is intended to help move the system towards meeting the yet-unfulfilled system requirements.

Evo steps are assumed to be small increments, typically a week in duration or 2% of total budget. There are two purposes for Evo steps: to move us towards the long-range requirements, and to learn early from stakeholder experience (with a view to changing plans and designs early).

Notes:

1. Evo Step Content: A step will contain the 'means' for meeting specified 'ends' (requirements). It will contain some combination of design ideas, which aim to achieve the requirements.
2. Dynamic Step Sequencing: Evo is conceptually based on the Plan-Do-Study-Act cycle. An Evo plan for a project consists of a planned series of Evo steps sequenced in order for delivery. Step sequencing for delivery can be roughly sketched or planned in actual time sequence. Step

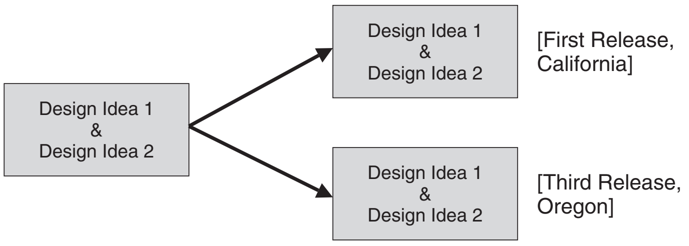


Figure G8
Evo Step packaging. Delivery-specified Evo Steps: same underlying step specification, but two different times and places.

sequencing always remains finally contingent upon actual feedback results, and on external considerations such as new requirements, changing priorities or new technology. The step delivery sequence is determined dynamically, after the previous step results are analyzed (Study Phase) and a decision is made about the next step (Act Phase). Selecting the *next* step for delivery is the main focus of Evo planning activity.

3. Step Priority: Steps with the highest stakeholder value to cost ratios or performance to cost ratios ought to be scheduled for early delivery. Step dependencies have also to be considered.
4. Step Size: A step is typically, but not unconditionally, constrained to be between 2% and 5% of a project's total financial budget and total elapsed time. Why? Well, because 2% to 5% is a reasonable amount of resource to gamble, if you are not absolutely sure whether a step will succeed.
5. Step Specification: An evolutionary step specification is the written description of the step content; that is, specification of the list of design ideas involved.
6. Step Lifecycle Location: A step is developed and delivered within a result cycle: any necessary step development occurs as part of the development cycle, any step production required occurs as part of the production cycle, and step delivery takes place as part of the delivery cycle.
7. Step Content Reuse: It is possible for the same step *content* to be repeated in several *different* steps (that is, in effect a 'roll-out' across a system, over time, such as to different countries or states or branch offices). In such cases, the step specifications will differ only in the qualifiers. For example, Step 1: Function XX [California], Step 2: Function XX [New York], and so on.

   Example:
   S23: Step [<Time, Place, Event to be determined>]: F1, F2 [Europe], D3 [China].
8. The main difference between an Evo step package (above example with undetermined qualifier) and a delivery-specified Evo step is that the latter has been assigned a sequence or timing and a place of application qualifier. The Evo step package is just a specification of the step contents. An Evo step package can be deployed in multiple times and places. You could say that an Evo step package is a reusable specification. For example, every week it could be to different countries.

Synonyms: Step *141; Evolutionary Step *141; Build, Increment, Installment, Release: Near synonyms depending on whether there is use of feedback and dynamic change (Royce 1998).
Related Concepts: Evo Step Specification *370; Evo Plan *322; Evolutionary *196; Evolutionary Project Management (Evo) *355; PDSA Cycle *168: See also the individual Plan, Do, Study, Act components; Result Cycle *122; Development Cycle *413; Production Cycle *407; Delivery Cycle *049; Result *130: Synonym is Step Result.
Keyed Icon: ->☺ or ->:) "Symbolizing 'Impact' on stakeholder. The ☺ symbol is sometimes automatic 'correction' for the colon and right parenthesis keyed symbols, in Microsoft Word."

Evolutionary — Concept *196

The 'evolutionary' concept implies association with Evolutionary Project Management; an iterative process of change, feedback, learning and consequent change. Evolutionary processes needs to be carefully distinguished from other processes – those that do not iterate, do not learn from experience, and do not cater for change.
Abbreviation: Evo *196.
Related Concepts: Evolutionary Project Management *355; Evo Plan *322; Evo Step *141; Plan-Do-Study-Act Cycle *168.

Evolutionary Project Management — Concept *355

A project management process delivering evolutionary 'high-value-first' progress towards the desired goals, and seeking to obtain, and use, realistic, early feedback.
Key components include:

- frequent delivery of system changes (steps)
- steps delivered to stakeholders for real use
- feedback obtained from stakeholders to determine *next* step(s)
- the existing system is used as the initial system base
- small steps (ideally between 2%–5% of total project financial cost and time)
- steps with highest value and benefit to cost ratios given highest priority for delivery
- feedback used 'immediately' to modify future plans and requirements and, also to decide on the next step
- total systems approach ('anything that helps')
- results-orientation ('delivering the results' is prime concern).

Description: Chapter 10, "Evolutionary Project Management: How to Manage Project Benefits and Costs".
Abbreviation: Evo *355.
Synonyms: Evo Management *355; Evolutionary Delivery Management *355; Rapid Delivery Management (Acronym: RDM), Result Delivery (These synonyms are used within Jet Propulsion Labs. (Spuck 1993); Synch-and-stabilize or Milestone Approach (These synonyms are used within Microsoft (Cusumano and Selby 1995); None of them are perfect synonyms, but since each author and company has a long list of extremely similar features that make up these processes, they are close enough.
Historical Note: Evolutionary Project Management had an early large-scale documented use in the Cleanroom techniques used by Harlan Mills within IBM in the 1970s (Mills 1980). Larman and Basili (2003) gives a comprehensive history of the method.

Except **Concept *389**

'Except' is used to specify that the following term or expression is an exception from the previous term or expression.

Example:
Goal [Europe Except {Denmark, France, Luxembourg}]: 20%.

Fail **Concept *098**

'Failure' signals an undesirable and unacceptable system state. A Fail level specifies a point at which a system or attribute failure state begins. A single specified number (like Fail: 90%) is the leading edge of a Failure Range.

A Fail parameter is used to specify a Fail level constraint; it sets up a failure *condition*.

Notes:

1. Failure ranges can be arbitrarily stipulated by a stakeholder. They might be stated in a contract. They are specified so as to keep designers and implementers aware of the levels at which stakeholders are likely to experience failure, or to contractually declare some degree of failure.
2. A failure *range* maps an extent of unacceptable levels. The failure range is better than 'catastrophic' levels, and worse than 'acceptable' levels. In other words, the failure range extends from the defined Fail level in the direction of 'worse' until a Survival level (or Catastrophe level) is reached.
3. The purpose of the 'Fail' concept is to inform us that we need both to design for, and operate at, more acceptable levels.

   Example:
   Fail [Euro Release]: 99.5%.

   For example, a state of failure can result from issues such as safety problems, operator discomfort, customer discomfort, loss of value, and loss of market share. Failure levels cause problems, even temporary system loss, but they are not immediately critical to a system's continued survival. The assumption is that it is possible to get the system out of a failure range.
4. Fail levels do not represent *total* failure. That role is defined by catastrophe levels. However, system development should keep going until, at least, the actual system levels are better than the specified failure levels. Otherwise, they are delivering some degree of failure to some stakeholder; that is, the system or attribute will at some stage fail in some sense.
5. A Fail constraint specification means that some defined stakeholder has stated the level at which the attribute's numeric value becomes unacceptable to them. Any level equal to or worse than the Fail level, is outside the 'acceptable' range for that stakeholder.
6. A systems engineer should document *why* a specific Fail level was chosen (using Rationale or similar), and the likely impacts (using Impacts) and consequences of any failure (using Risks), so that risk analysis and prioritization can be carried out.

   Example:
   Learning Time:
   Scale: Mean Time to Learn defined [Task] by defined [Operator].
   Fail [Outgoing Call, Beginner]: 3 minutes <- Marketing Requirement 3.4.5.

Risk: If the Mean Time is not lower, then Competitor Products will be perceived as better and we will lose <market share> <- Marketing Planner [Andersen].
Fail [Address List Update, Professional User]: 30 seconds <- Marketing Requirement 3.4.6.
Authority: External Consultants. "Outside consultants tell us we will be rated badly if we fail to beat this level."
Goal [Average Task, Average User]: 25 seconds <- Marketing Requirement 3.4.7.
Rationale: Marketing believes this will make us best in the Market.
The local parameters, Risk, Authority and Rationale can be used to explain why scalar levels have been set at specific levels. Note that the Source(s) of information (format: 'B <- Source of B') give indirect authority for the specification levels. (The Goal specification is included here to give a more realistic specification example.)

Synonyms: Fail Level *098; Fail Limit *098; Failure Level *098; Failure *098; Warning *098; Must (Avoid) *098: Historical usage only.
Related Concepts: Survival *440; Catastrophe *602; Range *552: See 'Failure Range'; Must Do *539: Historical usage only.
Keyed Icon: ! "In context on scalar arrows: ---!--->O---!--->
A Failure Range would use multiple Fail icons: ----!!!!!!--->-> "

Frontroom — Concept *343

Frontroom is an adjective or noun, referring to a conceptual place, used to describe any project management processes or activities, in Evo, that are visible to the Evo step recipients.

Notes:
1. Typically, 'frontroom' is used to refer to the delivery cycle part of the result cycle. The frontroom is where the step is delivered to the stakeholders.
2. The frontroom is where stakeholder-level results of the step integration can be tested and measured.

Related Concepts: Backroom *342.

Function — Concept *069

A function is '*what*' a system does.
A function is a binary concept, and is always expressed in action ('to do') terms (for example, 'to span a gap' and 'to manage a process').

Notes:
1. A function has a corresponding implied purpose. For example 'to span a gap' usually has as an implied purpose to enable something to get from point A to point B over the gap.
2. Function is a fundamental part of a system description: a system consists of function attributes, performance attributes, resource (cost) attributes and design attributes. All attributes exist with respect to defined specified conditions.
3. All the system attributes must be described together, in order to fully understand a real world system. Function is a 'pure' concept, which cannot exist in the real world alone. Functions need to have associated with them, the relevant performance and resource attributes.

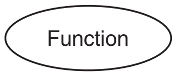


Figure G9
The drawn icon for Function *069.

Further, functions can only exist and successfully interact with the real world if they have certain minimal levels of such attributes, for example, levels of availability.

4. Function is a system attribute that is expressed without regard to the related performance, cost and design. A function needs to be clearly distinguished from design ideas. Design ideas are a real world method for delivering all the function, performance and cost attributes of a system.
5. Function itself is binary. Function in a system is either implemented or not. It can be tested as present or not. Any *real* implemented function will have some associated performance and cost attributes – whether we planned for them or not. Once a design with the required functionality is specified (or later implemented), we need to consider whether that particular design has *satisfactory* performance and resource (cost) attributes. In other words, we control these scalar attribute's levels mainly by specifying appropriate design options, which deliver the required performance and cost levels.
6. A function can often be decomposed into a hierarchical set of subfunctions. For specification clarity, prefixes such as 'sub-', 'supra-' or 'family' relationships (such as kid, parent, sibling) can be used to express the relationships amongst the different functions. Alternatively, the parameters, Includes, Is Part Of and Consists Of can be used.
7. I have intentionally chosen the term 'function' as the adjective for 'function' (for example, in 'function specification' and 'function requirement'), rather than the more common 'functional.' It is the 'requirement for function' that is being expressed, rather than 'making the requirements functional.' The logic of this choice is the same as for choosing 'quality' (for example, in 'quality requirements), rather than 'qualitative.'

Related Concepts: Attribute *003; Design Idea *047; Mission *097; Function Design *521.
Drawn Icon: An oval (a circle would also be considered a function.)
Keyed Icon: O or parentheses, ( ) "In context: ------->O------> This describes a system: the function keyed icon, 'O,' is combined with two scalar arrows representing scales of measure for cost and performance attributes. Alternatively: ----->(<*function tag*>)----> "

Function Constraint — Concept *469

A function constraint is a requirement, which places a restriction on the functionality that may exist in a system.
A function constraint is binary: it specifies that a specific function must be, or must not be, present. The implication is that some kind of failure will result if a function constraint is not met (such as contract penalties).

Example:
No New Games:
Type: Function Constraint.

Rationale: No new games of any kind will be available on the new product.
Definition: No functionality required solely for a New Game is to be developed.
Support for Old Games [Release 1]:
Type: Function Constraint.
Rationale: All available games from our older product will be available to any customer on request. Some customers would be upset at losing the existing games.
Definition: Functionality to support Old Games must be included.
Related Concepts: Function *069; Function Target *420; Requirement *026.

Function Design — Concept *521

Function design is a design primarily aimed at satisfying specified function requirements. A specified function design has two characteristics, which we primarily select it for:

- function requirement satisfaction, and
- satisfactory consequent levels of performance and cost attributes.

Example:
Cross River:
Type: Function Requirement.
Definition: Move people and goods from one shore to the opposite shore of a river.
Function Design Ideas [Version 1]: {Build a Bridge, Use a Boat, Swim Over "minimal design," Take Route 'Around' the River, Fly Over}.
Consideration of potential design ideas: Function Design Ideas [Version 1] shows selecting on function satisfaction: some function designs, which satisfy the function requirement.

Example:
Function Design Ideas [Version 2]: {{Build a Bridge and/or Use a Boat}, Not {Swim Over, Take Route 'Around' the River or Fly Over}}.
Function Design Ideas [Version 2] shows further selection using knowledge of performance and resource (cost) attributes: The function designs that look most promising for the system.

Notes:

1. The final real performance and cost levels delivered is dependent on the *specific* design chosen (for example, exactly what specific *design* of bridge, or specific *type* of boat).
   Functional design necessarily narrows the *remaining* design scope to some degree. It can even narrow the design scope to a set of function designs *without* actually taking a *final* choice of specific design (for example, Build a Bridge and/or Use a Boat – without yet saying exactly which type). The final design specification would then be left to a downstream design process.
   This delay might be justified by their more specialized knowledge downstream, or justified by the advantage of putting off the decision due to changed technology/market conditions/costs, or due to an advantage of making the decision in the context of many other system/project-wide decisions (avoiding sub-optimization).

2. Any function design in its real implementation (as opposed to pure function specification) will impact many of our non-function requirements (performance requirements, resource (cost) requirements, condition constraints or design constraints). This multiple impact is inevitable whether we like it or not. We cannot, it seems, only design for one pure requirement dimension without having some effects on the others.

   When a design is primarily specified for *non-function* purposes (like improving a quality level), it might inadvertently impact existing functionality as a side effect. This might possibly be acceptable. It might introduce new function, modify old function or make existing function inaccessible totally or practically.
3. The key reasons for considering function design, as a distinct design type, is that:
   - you can narrow the function design scope gradually
   - you become more conscious of the side effects on performance and cost
   - you become more conscious of the necessity of choosing function design alternatives on the basis of their impacts on performance and cost.
   - you can separate the design rationale for the function, from consideration of the other attributes.

Related Concepts: Function *069; Function Requirement *074; Design Process *046; Design Specification *586.

Function Requirement — Concept *074

A function requirement specifies that the presence or absence of a defined function is required.

A function requirement is binary, and can either be a specific function target or a generic function constraint.

Example:

Voice Recognition:

Type: Function Requirement.

Definition: The ability to recognize a human voice in terms of vocabulary and individual voiceprints.

Step 1: Step: Voice Recognition [Europe, If Company C has this function on the market].

Voice Recognition is defined as a function. It is then 'required' to be delivered in Evo step 'Step 1,' only in 'Europe' and only if 'Company C has this function on the market.' A specific design to implement Voice Recognition needs to be specified.

Notes:

1. Do not include technical design ideas in function requirements. Designs are quite different from functions. If designs are mandatory, then they should be specified as design constraints. A function is an abstract concept specifying activity of some kind, which is implemented by a design. For example: An accounting application (a design) provides a solution to support Maintaining Accountancy Information (a defined function).
2. Distinction should be made between a function target and a function constraint. A function constraint implies that a function must be

present or absent (subject to its qualifier conditions) in a system, or a penalty of some kind will be incurred.

3. By default, if there is no information that a function requirement is actually a function constraint, or 'Type: Function Constraint' specification, a function requirement is assumed to be a function target.
4. To authority levels, which are lower than the one that specified it, a function target does become mandatory. If a lower authority disagrees with a requirement they have to take the issue up with the higher authority.
5. A function requirement is satisfied by any design, which meets the function description. For example, {transport via a bridge, transport by air, transport across water} all meet the function requirement 'to transport people from shore to shore of a river.' Note, in this example, the designs are high-level and are actually functions. They can be termed 'function designs.' A lower, more specific level of design {by public transport over a bridge, by hot-air balloon, by canoe} can also be considered.

   At the early rough stages of design, function requirements are best satisfied by rough function designs (like 'bridging the river'). At the latter stages of design, specific designs are better, like 'rope bridge.' The issue is that the more specific a design is, the less freedom of design choice remains, but the greater the knowledge of its attached performance and resource attributes. For example, the quality attributes of a software package selected to satisfy the function requirement, like reliability and portability.

Synonyms: Functional Requirement *074.

Related Concepts Function Target *420; Function Constraint *469; Function Design *521; Function *069.

Function Target — Concept *420

A function target is a specified function requirement. We need to plan delivery of the function under the specified conditions.

A function target can be contrasted with the other class of function requirement, a function constraint. A function constraint specifies mandatory functionality (either a function has to be present or absent), as a penalty of some kind will result if the constraint is not met.

Example:
Propulsion Capability:
Type: Function Target. "Could also be termed a Function Requirement."
Description: A means to mechanically drive the vehicle around in three dimensions.
Basic definition of a function target.

Example:
Step 22:
Type: Evo Step.
Dependency: Step 23 completed successfully.
Step Content: Propulsion Capability [Version = Prototype, Capability = Surface Movement, Means = Electrical-Powered].
Exploitation of the function target specification by referencing its tag in an Evo step plan, with suitable qualifiers. Notice how the qualifiers make the generic function somewhat more specific.

Related Concepts: Function Constraint *469; Function Requirement *074; Target *048; Function *069.

Fuzzy **Concept *080**

A specification, which is known to be somewhat unclear, potentially incorrect or incomplete, is called 'fuzzy.' It should be clearly declared as 'fuzzy.'
The keyed icons '< >' are used to explicitly mark any fuzzy specifications.

Example:
Scale: <define units of measure>.
Note: This is a template with a hint in fuzzy brackets.
Goal [<Europe>, <2005>]: <66%>.

Rationale: The idea is to avoid forgetting to improve specifications, and to avoid misleading other people into thinking you have done your potential best, when you know better should be done, when you have time and information, in order to define specifications at the necessary quality level.

Notes:
1. The obligation to mark dubious specifications with fuzzy brackets is typically adopted as a generic specification rule.
2. In general, a fuzzy term or expression should be enclosed in <fuzzy brackets>, but alternative notations (such as '??') can also be used.
3. Fuzzy brackets are used in electronic templates to indicate something to be filled out, and usually to give a hint as to what should be filled out. See Scale in example above.
4. A fuzzy specification essentially amounts to a declaration by the writer that the specification is defective at that point.

Keyed Icon: <*fuzzy term*>

Gap **Concept *359**

For a scalar attribute, a gap is the range from either:
• an impact estimate, or a specific benchmark (usually the current level),
• to a specific target (or occasionally, to a specific constraint).

Notes:
1. In general, the larger the gap, then the greater the need to deal with it ('the higher the priority') in order to reach the target or constraint. Of course, large gaps could be easy and some small gaps could be difficult, so that is why this paragraph says ' in general.'
2. When a gap no longer exists for a specific scalar attribute, then that attribute ceases to have 'claim on project resources' (priority). It then has no priority.

Related Concepts: Range *552; Design Problem *048.

Gist **Concept *157**

A Gist parameter is used to state the essence, or main point, of a specification. A Gist is a summary of the detailed specification.

Notes:
1. A good Gist serves two purposes:
   • it helps a planning group to agree on the summary of a specification, before they spend more time formulating the specification in greater detail.
   • it summarizes a detailed specification. This serves several purposes:
     – readers can quickly grasp the subject matter
     – readers can decide to avoid the detail

- presenters can refer to a specification by using just its tag name and Gist. (A reader who needs more detail can 'drill down' to the detail using the tag name.)

Example:
Gist: All the functions related to transportation of people.

Example:
Software Interfaces:
Type: Architecture.
Σ: The total set of software interfaces needed for this project.

2. The detailed specification may already exist, or it may be made on the basis of an agreement about the Gist.
3. When summarizing a *scalar* specification, use the more specific parameter 'Ambition.'

Synonyms: Summary *157.
Related Concepts: Ambition *423.
Keyed Icon: Σ "Greek Summa, mathematical summary symbol."

Goal — Concept *109

A goal is a primary numeric target level of performance. An implication of a Goal specification is that there is, or will be, a *commitment* to deliver the Goal level (something *not* true of a Stretch or Wish target specification). Any commitment is based on a trade-off process, against other targets, and considering any constraints. The specified Goal level may need to go through a series of changes, as circumstances alter and are taken into consideration.

A specified Goal level will *reasonably satisfy* stakeholders. Going beyond the goal, at the cost of additional resources, is not considered necessary or profitable – even though it may have *some* value to do so.

A Goal parameter is used to specify a *performance* target for a scalar attribute.

A Goal level is specified on a defined scale of measure with its relevant qualifying conditions [time, place, event].

Notes:

1. To reach a Goal level is a *success* to specific stakeholders. It is also a sort of 'stop' signal (a red light) for use of project resources on the specific performance attribute concerned: although better levels might be reached, and might be of value to some, they are not called for, under the stated conditions. For example, the additional value gained, given the estimated costs, is not viewed as worthwhile. In economic terms, we have at the Goal level probably reached the point of diminishing return on investment.
2. 'Goal' is intentionally not used for *resource* targets ('Budget' is used instead).
3. I now prefer the term 'Goal' instead of my traditional 'Plan' parameter. 'Plan' refers to so many other elements of planning. If an alternative were needed for Goal, I would use the more explicit 'Planned Level.'

Example:
Glory: "Humpty's and Alice's problem, what does 'glory' mean?"
Scale: Number of Literature Citations to a defined [Person's Work] during a defined [Time Span].
Goal [Person's Work = The Academic, Time Span = Each Decade]: Over 1,000 <- Prof. H G. "That is glory!"

Synonyms: Plan *109: Historic usage only; Planned Level *109: Historic usage only; Goal Level *109: See Level *337; Planned Goal *109.
Related Concepts: Aim *001; Target *048; Stretch *404; Wish *244; Ideal *328; Objective *100.
Keyed Icon: > "A single arrowhead, on a performance arrow, pointing towards the future. It is the same icon as for Budget *480 (which is on a resource arrow, --->--->O).
In context: O---->---->
Always use an output arrow from a function oval to represent a performance attribute. The Goal icon is the '>' on the scalar arrow. If other scalar levels are shown, the positioning of the tip of the icon symbol should reflect the Goal level relative to these other levels."

Icon **Concept *161**

In Planguage, an icon is a symbol, that is keyed (Keyed Icon) or drawn (a Drawn Icon), that represents a concept. All icons are graphic or pictorial in nature – they should not use words or national languages.
Related Concepts: Keyed Icon *144; Drawn Icon *085; Symbol *161.

IE **Concept *283**

Acronym for 'Impact Estimation'.

If **Concept *399**

'If' is a logical operator used in qualifiers to explicitly specify conditions.
Notes:
1. The 'If' is *implied* for all terms in a qualifier. However, 'If' may be used to communicate a condition more explicitly to the novice reader.
   Example:
   Goal [USA, If Law 153 Passed]: 99.9%.
   Goal [If Europe, If Product XYZ Announced]: 60%.

Synonyms: IF *399.
Related Concepts: Qualifier *124; Condition *024.

Impact **Concept *087**

An 'impact' is the estimated or actual numeric effect of a design idea (or set of design ideas or Evo step) on a requirement attribute under given conditions.
Notes:
1. Full impact information includes the following: a scale impact, a percentage impact and uncertainty data (known error margins). The additional related information required to support an impact includes the evidence, source(s) and credibility.
2. If an impact is estimated, it is an Impact Estimate *433.

Related Concepts: Impact Estimate *433; Impacts *334.

Impact Estimate **Concept *433**

An impact estimate is an evaluated guess as to the result of implementing a design idea. In other words, it is a considered, quantified guess of the effect on a specific scalar requirement attribute (performance or resource) of implementing a design idea (or set of design ideas) in a system (or system subset) under stated conditions.

A full impact estimate includes the following: a scale impact, a percentage impact and uncertainty data (known error margins). The additional related information required to support an impact estimate includes the evidence, source(s) and credibility.

Notes:

1. An impact estimate can be positive, neutral or negative (undesirable in relation to stated target levels).
2. Note the distinction between a scale impact (an absolute numeric value on a Scale), and a percentage impact (the percentage improvement estimated to be achieved in moving from the chosen baseline towards the chosen target).
3. An impact estimate is usually concerned with the system improvement, rather than with stakeholder value (however, this depends on the choice of requirement attribute: stakeholder value can be tackled, for example, Financial Saving).
4. An impact estimate is usually on a scalar requirement. However, much more rarely, it can be on a binary requirement of the system (that is, on a function requirement, a design constraint or a condition constraint). This is used in situations where an explicit check is considered necessary to help evaluate design ideas.

Abbreviations: Impact *433 "Often 'Impact' is short for 'Impact Estimate'. See also Impact *087."

Related Concepts: Impact *087; Scale Impact *403; Percentage Impact *306; Side Effect *273.

Impact Estimation **Concept *283**

A Planguage method/process used to evaluate the quantitative impacts of design ideas on requirements.

Description: Chapter 9, "Impact Estimation: How to Understand Strategies and Design Ideas."

Acronym: IE *283.

Historical Note: History of the development of Impact Estimation: I've developed the IE method in the course of my consultancy work. I originally started in the early 1960s with multidimensional evaluation models, which were later published (in 1968 at the Nord Data Conference) as 'Weighted Ranking by Levels' or the MECCA method. By the 1970s, I had adopted a table format. This was chiefly inspired by the 'Requirements/Properties Matrix' presented by Dr. Barry Boehm, then of TRW Systems, in his 1974 IFIP Speech in Stockholm. The main difference in my approach was that I wanted to provide a more quantified method; while I liked the idea of the matrix structure, I found the TRW implementation too fuzzy. See my 'Software Metrics' book (Gilb 1976 Out of Print). Finally, by the mid-1980s, I had the basics of Impact Estimation (IE), which was described in my Principles of Software Engineering Management (Gilb 1988) book. Subsequently, during the early 1990s, we (Kai Gilb and I) added credibility evaluation and the use of graphical 'skyscraper' representation. I have also recently started using IE to explicitly outline evolutionary step sequences.

Notes:
Key Differences: Impact Estimation and QFD. I am frequently asked to compare IE with Quality Function Deployment (QFD) (Akao 1990). The key difference between IE and QFD tables lies in *the degree of quantification*. In QFD, the objectives are rarely stated quantitatively, design ideas tend not to be formally specified, cost is hardly ever considered and the evaluation of the impact of the design ideas is not numeric (usually only an assignment of 'weak', 'medium' and 'strong' is made). Also, there is no attempt at citing evidence, sources or credibility.

Impacts **Concept *334**

The 'Impacts' parameter is used to identify the set of attributes that are considered likely to be impacted by a given attribute (usually another requirement attribute or a proposed design idea).

Notes:
1. 'Impacts' can be used to capture Impact Estimation table relationships before actual numeric estimation.
2. 'Impacts' differs from 'Supports' in that it can be used to identify *side effects, including negative side effects*, as well as the intended direct positive impacts.

   Example:
   Design Idea 1: Handbook Impacts {Learning, Development Cost}.
   or
   Design Idea 1: Handbook -> {Learning, Development Cost}.

Keyed Icon: ->
"The 'Impacts' arrow is only valid in the context of tags referring to things that can impact one another."

Example:
Design A -> Requirement B.

Related Concepts: Supports *415; Is Impacted By *412.

Includes **Concept *391**

'Includes' expresses the concept of inclusion of a set of components within a larger set of components. 'A Includes B.' means that B is a sub-component of the component A.

Example:
B: Includes {C, D, E}.

Example:
Bee.Wings "Wings is a member of supra concept, or parent concept, Bee."
Bee: Includes {Wings, Legs, Eyes, Sting, Body}.
Bee: {Wings, Legs, Eyes, Sting, Body}. "Includes is implied by the set parenthesis."
Alternative formats and synonyms for Includes.

Related Concepts: Consists Of *616.
Keyed Icon: { } "In context, X: {A, B} means X includes A and B."

Incremental Development **Concept *318**

Incremental development means designing a system largely up-front, and then dividing its construction, and perhaps handover, into a series of cumulative increments.

Notes:

1. Incremental Development is defined here in order to contrast it with, and distinguish it from, Evo:
   - Incremental development differs from Evo in that most all of the Incremental Development requirements design effort is up-front. In contrast Evo carries out requirements and design detailing *gradually* in each Evo cycle
   - Incremental development is without Evo's intent of *measuring* the progress of each (incremental) step fully (for example, measuring delivered performance levels), then *learning* from these feedback measures and, *changing* the requirements and/or design accordingly
   - Incremental development is also without the intent of delivering the steps (increments) with the highest 'value to cost ratio' or 'performance to cost ratio' first.
2. It is unfortunately common practice to say or write 'incremental' when the strictly correct term according to the distinctions defined here is 'evolutionary.' Indeed, all evolutionary processes are also incremental, but they are a subclass deserving distinctive terminology to announce the differences. This 'lazy' use of the term is a sure sign of people who do not have deep understanding, or concern for, the value of feedback and change. Beware of their advice or opinions! The US DoD (DoD Evo 2002 http://www.acq.osd.mil/dpap/ar/1_multipart_xF8FF_2_EA%20SD%20Definitions%20final.pdf), among others, has taken the trouble to carefully distinguish these concepts!

Related Concepts: Evolutionary Project Management (Evo) *355.

Incremental Scale Impact — Concept *307

For a scalar requirement, this is the numeric impact of a design idea *relative* to the specified baseline level. If there is a negative impact, then the numeric value will be negative.

Scale Impact – Baseline = Incremental Scale Impact

Example:

Consider an objective concerning say, a 'Customer Response Time,' with a defined Scale of 'Minutes to Wait.' If the Baseline was 'Past: 20 minutes to wait' and the Target was 'Goal: 5 minutes to wait' and, the Scale Impact (estimated or actual) of Design Idea X on Customer Response Time was a result of '12 minutes to wait,' then the Incremental Scale Impact is 8 minutes (20 − 12 = 8).

The Percentage Impact is 8/15 or 53% relative to the Baseline (0%, or 20 minutes) and to the Target (100%, or 5 minutes).

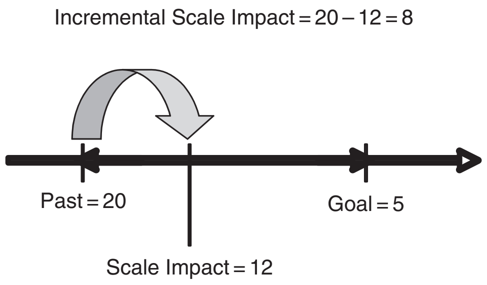


Figure G10

Notes:

1. Designs vary in their impact, depending on previous circumstances. The incremental impact is a function of these circumstances. The impact of a design idea is *not a constant*, irrespective of the circumstances it is implemented in.

Related Concepts: Percentage Impact *306; Scale Impact *403.

Inspection — Concept *051

'Inspection' is a synonym for Specification Quality Control (SQC).

Notes:

1. Michael Fagan originated the term 'Inspection' in connection with software within IBM. He developed the initial method for quality control of software. It is based on the work of Walter Shewhart, Joseph Juran and others, who used the term for quality control of products (rather than of specifications). Given the confusion in engineering environments over the use of the term 'Inspection' (to hardware engineers it means quality control after production of something), I prefer to use the term, SQC.
2. Many people incorrectly equate the Defect Detection Process (DDP) with 'Inspection.' They omit the Defect Prevention Process (DPP). This is because they are unaware of the additional developments to Inspection introduced by Mays and Jones (Mays 1995).

Synonyms: Specification Quality Control (SQC) *051; Peer Review *051.

Is Impacted By — Concept *412

'Is Impacted By' is used to indicate any *other* specified items (such as requirements, objectives, designs, policies or conditions), which affect, or might affect, a defined specification itself, or what it refers to.

Notes:

1. The purpose of 'Is Impacted By' is to help in risk identification and analysis. We are trying to explicitly identify and document factors, which we believe influence the results. This will hopefully result in specific action or design to keep those impacts from threatening our planned results.
2. The more general purpose of Is 'Impacted By,' and many other Planguage relationship mechanisms, is to build a 'web of connections' between specifications (that is, between system components). This web of connections serves many purposes. Risk management was mentioned above. Other uses are configuration management, system familiarization, quality control, estimation, contracting, prioritization, and reviewing.

   Example:
   A:
   Is Impacted By: {Help Desk Capacity, User Motivation, User Training, Bug Frequency}.
3. 'Is Impacted By' is differs from considerations of Risk/Threat in that both *good* and bad impacts are considered. With Risk/Threat, we are primarily concerned with the potential for *negative* impacts.

4. For strong primary intended impacts, the 'Is Supported By' icon can be used, A ->> B. meaning B is supported by A. In other words, A is *primarily* the way we intend to achieve the requirement/value B.

Synonyms: Impacted By *412.

Related Concepts: Risk *309; Threat *309; Dependency *189; Impacts *334; Supports *415; Is Supported By *414.

Is Part Of **Concept *621**

'Is Part Of' is a parameter, which indicates that a specification is a component or element of some other component or element.

Example:
PDSA Cycle:
Type: Process.
Consists Of: Sub-Process {Plan, Do, Study, Act}.
Plan:
Type: Sub-Process.
Is Part Of: PDSA Cycle.

Example:
Reliability Is Part Of Availability.

Related Concepts: Consists Of *616; Includes *391; Component *022; Element *022.

Is Supported By **Concept *414**

'Is Supported By' is used to list the tags of any and all attributes that contribute usefully to the accomplishment of the planned target levels of a defined requirement.

Notes:

1. The attributes that can provide support include designs and Evo steps.

   Example:
   Goal X:
   Scale: <some scale definition>
   Goal [Next Version]: 55%.
   Is Supported By: Design Idea {A, B, C}.

Synonyms: Supported By *414.

Related Concepts: Supports *415; Impacts *334; Is Impacted By *412.

Issue **Concept *276**

An issue is any subject of concern that needs to be noted for analysis and resolution.

Example:
ISS1: Issue: We have not analyzed risks and dependencies yet.

Notes:

1. A specification issue is an element of written specification, which we suspect violates a specification rule. It is noted for later resolution. It will be resolved by being declared to be either a defect (a rule violation) or as requiring no further action.

Related Concepts: Resolution *525; Specification Issue *529.

Kin **Concept *353**

Kin specifications are specifications, which derive from an identical set of source specifications.

Notes:

1. For example, test plans, source code and user handbooks could all be derived from the same requirements or the same design.
2. Kin specifications can serve as additional information to perform defect checking in the Specification Quality Control (SQC) process. For example, United Defense in Minnesota reported [personal communication] that their software engineering checked the program code against their test cases, both derived from the same requirements. They reported that they usefully found major defects in both these kin documents.

Landing Zone **Concept *605**

A landing zone is a target range that stretches from just better than a Fail level through the Goal/Budget level to the Stretch Level.

Notes:

1. A landing zone is analogous to a parachute's landing zone. A range that we realistically hope we can land in somewhere. This avoids the simplified notion of an exact Goal/Budget being the target.
2. For a set of requirements, the overall landing zone is the set of landing zones, which 'creates a space' over all the requirement dimensions.
3. The multidimensionality of landing zones is an important feature. The space below Goal may seem unacceptable, but when you consider all dimensions at once, sub-par achievement in a single dimension is completely acceptable, if it means optimal system performance.
4. A landing zone covers a success range and an acceptable range.

Related Concepts: Range *552.

Historical Note: The source of the Landing Zone concept was Intel, Oregon (via Erik Simmons, 2002).

Table G2 A simple example showing multidimensional landing zones. It is landing within all the landing zones simultaneously that is the aim. A teaching example using fictional data. (Courtesy of Erik Simmons, Intel).

Attribute	*Fail*	*Goal*	*Stretch*
Price	>$27000	$20000	$17,500
Mileage (City)	<18 mpg	25 mpg	35 mpg
Seating	<4 adults	5 adults	6 adults
Interior Noise at 65 mph	>74 dBA	65 dBA	55 dBA
Projected 3-year Maintenance Cost	>$3000	$2000	$1500

Level **Concept *337**

A level is a defined numeric position on a scale of measure.

Notes:

1. A scalar level applies to either a performance or a resource attribute.
2. A level on a scale of measure indicates one of the following:
   - a benchmark: an actual measurement or estimated level in the past
   - a target: a requirement level
   - a constraint: a limit
   - an estimate of the impact of a design idea

Synonyms: Point *337: A position on a Scale.

Related Concepts: Range *552; Goal *109: Goal Level; Budget *480: Budget Level; Stretch *404: Stretch Level; Wish *244: Wish Level; Fail *098: Fail Level; Survival *440: Survival Level; Catastrophe *602: Catastrophe Level; Past *106: Past Level; Record *127: Record Level; Trend *155: Trend Level; Limit *606: An extreme boundary of the range of a level.

Keyed Icon: | "In context on a Scale: ----|---> This is the generic attribute level icon. It can be used instead of any of the more specific level icons (for example, '>' for Goal or Budget)."

Limit **Concept *606**

A limit is a numeric level at a border, that is, at an edge of a scalar range (a success range, an acceptable range, a failure range or a catastrophe range). It is specifically used at the edges of ranges associated with constraints: fail limit and survival limit/ catastrophe limit.

Related Concepts: Range *552; Fail *098: Fail Limit; Survival *440: Survival Limit; Catastrophe *602: Catastrophe Limit.

Logical Page **Concept *103**

A logical page is defined as a defined number of *non-commentary* words. Default Volume: If no other definition is given, use '300 non-commentary words' per logical page as default.

Rationale: This measure of specification 'volume' is used to make sure that varying page sizes and page content does not cause false volume measures. Volume measures are important for establishing checking rates (logical pages per hour) and defect density (majors per logical page).

Notes:

1. 'Non-commentary' is a useful concept because it only pays off to worry about optimum checking rates or defect densities on non-commentary specification (where potential danger lies in defects).

Abbreviations: Page *103: In an SQC context.

Acronym: LP *103.

Synonyms: Logical Page Size *103.

Related Concepts: Physical Page: This is one side, facing a reader, of space for textual and/or graphical symbols, of physical or electronic nature. It has clearly defined borders, traditionally rectangular, with any arbitrary quantity of symbols.

Major Defect **Concept *091**

A major defect is a specification defect (a rule violation), which if not fixed at an early stage of specification, then it's consequences will possibly grow substantially, in cost-to-fix and/or damage potential. A

major defect has on average approximately an order of magnitude more downstream cost potential than it's cost to remove immediately.
Rationale: This concept and classification is necessary to help SQC checkers and other QC people to focus on what defects it pays off to find and eliminate in a specification. Without this classification, up to 90% of QC effort might be wasted dealing with minor defects.
Abbreviations: Major *091; M *091: Often intentionally written with a capital 'M'.
Related Concepts: Specification Defect *043; Minor Defect *096.

Master Definition **Concept *303**

The master definition of a specification or specification element is the primary and authoritative source of information about its meaning. The master definition overrides any other (informal, not master) definition that is in conflict with it.
Notes:
1. This glossary contains master definitions, but the definitions themselves may contain explanations of other terms (for example, in the Related Concepts sections), which are less formal and less authoritative than the master definition for that concept.
2. A master definition should contain full information about the source, authority, version and status, where relevant.
3. It is good practice to only permit a single master definition for a term to exist, and all references concerning the master definition must point to that single definition.

   Example:
   Master Definition: The primary correct source of a term's meaning.
   Type: Master Definition.

Measure, To **Concept *386**

To measure is to determine the numeric level of a scalar attribute under specified conditions, using a defined process, by means of examining a real system.
Notes:
1. Measurement is done on the defined Scale, with respect to specified qualifier conditions.
2. Measuring is done using defined Meters.

   Example:
   Usability:
   Scale: Mean Time To Learn.
   Meter [Experts]: Use the upper 5% of our experienced staff in tests.
   Meter [Novices]: Use 10% of current year's intake of new people.
   Fail [Experts, Complex Task]: 15 minutes.
   Goal [Novices, Simple Tasks]: 10 minutes.
   Two different Meter specifications are made in order to make it clear how the two different targets shall be measured.
3. Measuring is distinct from quantification and estimation. Quantification is merely defining an attribute with the help of a scale of measure, and benchmarks and/or target values. Estimation is trying to determine results based on past data.

Related Concepts: Scale *132; Meter *093; Quantify, To *385; Estimate, To *059.

Meter **Concept *093**

A Meter parameter is used to identify, or specify, the definition of a practical measuring device, process, or test that has been selected for use in measuring a numeric value (level) on a defined Scale.

"there is nothing more important for the transaction of business than use of operational definitions."

(W. Edwards Deming, Out of the Crisis *(Deming 1986))*

Example:
Satisfaction:
Scale: Percentage of <satisfied> Customers.
Meter [New Product, After Launch]: On-site survey after 30 days use for all Customers.
Past [This Year, USA]: 30%.
Meter [Past]: Sample of 306 out of 1,000+ Customers.
Record [Last Year, Europe]: 44%.
Meter [Record]: 100% of Customers.
Goal [After Launch]: 99% <- Marketing Director.
In the above example, the first Meter specification is the one that will be assumed, in default of any other specification, particularly for use in validating the achievement of Goal targets. Both the benchmarks (Past and Record) have local Meter specifications, which tell us more exactly the measuring process used to gather their data. Of course, this implies that these benchmark and target numbers are not as comparable as we would like them to be. But that is the way it often is, and our local Meter specifications at least allows us to judge whether this difference is significant for our current purposes.

Keyed Icon: -|?| – "A '?' on top of a Scale icon, -|-|-."

Metric **Concept *095**

A metric is any kind of *numerically* expressed system attribute. A metric is defined in terms of a specified scale of measure, and usually one or more numeric points on that scale. The numeric points can be expressed with defined terms that can be translated into numbers. For example, 'Record +10%.'

Normally there will also be other parameters and qualifiers, which add background detail to the metric. For example, Meter and Assumption. A metric specification encompasses *all related elements* of specification, not just the Scale of the numeric attribute.

A complex specification, with a set of scales of measure, is also a metric expression. There is no implication that it is elementary (has only a single Scale).

Notes:

1. Metrics are used to express 'concepts of variability' clearly – in particular more clearly than mere words (Gilb 1976).

   Example: [Metric Expressions]:
   Scale: Mean Time Between Failures.
   Use: Scale: Time to Learn defined [Average Task]. Past: 30 minutes.
   Design A: Impacts Requirement B: 30%.
   Each of these 3 statements is based on a 'metrics culture.'

Example: [Non-Metric Expressions]:
"Reliability"
"Easy to learn"
"Very effective design"
Each of these 3 expressions is based on a 'non-metrics culture.' Nice words – no numbers expressed, defined or implied.

2. The rationale for using metrics includes:
   - to increase *clarity and unambiguousness* of specifications
   - to increase *sensitivity* to small changes in specifications and the system itself
   - to enable systems engineering *logical thinking* about relationships – for example the relation of designs to requirements
   - to provide a better basis for *legal contracting* about systems
   - to enable *evolutionary tracking* of progress towards goals
   - to enable a *process of learning* within projects, engineering and management domains
   - to force engineers to *think* more clearly and *communicate* more clearly with others
3. A metric can be used to express the numeric impact of a *design* on a performance requirement (for example, when using the Impact Estimation method).

Related Concepts: Scale *132; Meter *094.

Minor Defect — Concept *096

A minor defect is a non-major defect. It has no major downstream cost potential.

Notes:

1. A defect which, if not removed at a given time, can be removed later (for example, in test phases or in customer use) at approximately the same cost or penalty.
2. There is little value in dealing with it immediately after it occurs. It can be left to chance, or ignored until it surfaces of its own accord.
3. Minor does not refer to the size of the defect, but to the potential consequences of it downstream.

Abbreviations: Minor *096; m *096: Deliberately written with a small 'm'.
Related Concepts: Major Defect *091.

Mission — Concept *097

A mission specifies who we are (or what we do) in relation to the rest of the world. It is the highest level of function of a system. The mission should not contain 'vision' description ('We make the best planes in the world'). It is an undramatic statement of the main function of a business or organization.
Mission, like function, intentionally excludes specific levels of attributes in its description.

Example:
Mission: We make semiconductors.
Mission: We provide business solutions in manufacturing software.

Related Concepts: Function *069.

Non-Commentary **Concept *294**

'Non-commentary' refers to written specification that is not commentary: it is either core specification or background specification. All major specification defects are found in non-commentary. But, not all specification defects in non-commentary specifications are major. By definition, major defects *cannot* be found in 'commentary.'

Notes:

1. Text diagrams or symbols that are secondary to the main specification purpose, and which do not lead to 'real product' are 'commentary.'
2. Non-commentary sections of a specification can be termed 'meat,' and commentary sections can be termed 'fat'. Commentary includes {notes (like footnotes), comments ("like this"), remarks (Note), introduction, and references (Source)}.
3. Checkers, in SQC, should concentrate on carrying out rigorous checking, at optimum checking rates, on the non-commentary territory. This gives better efficiency in finding defects.
4. It is important to formally distinguish between non-commentary and *commentary*. Authors/writers need to try to make the distinction visually for readers (for example, by using plain text for non-commentary and *italics for commentary*). When the visual distinction is made, and it is clear what is commentary and what is not, then quality control analysts can more easily, and more certainly, decide which defects are major and which are not. They can more quickly scan the commentary and more carefully study and cross-reference check the core specification, and then, somewhat faster, the background specification.

Related Concepts: Commentary *632; Background *507; Core Specification *633; Specification *137; Major Defect *091.

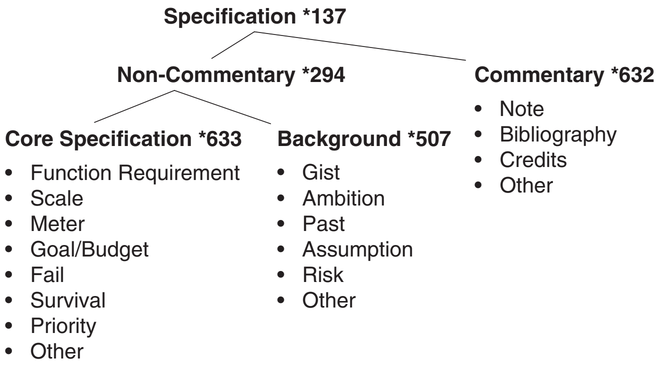


Figure G11
The diagram shows the relationship amongst the different categories of specification.

Note **Concept *018**

A 'note' is a comment or any text that makes any kind of remark related to any statement.

Ways of specifying notes include: *italics*, the use of ''quotation marks'', and the Note parameter (which has a synonym of 'Comment').

Example:
Ambition: Main share of the market. "This is just an example of a comment using quotes in a background statement."
Note: This is an example of the use of the Note parameter.

Notes:

1. Notes must be distinguished from the 'significant' core specification (for example, Goal and Scale) and from 'background' specification (for example, Source, Evidence and Gist). The main reason for this being that a defect in Note specification is usually only a minor defect. Any SQC checking should concentrate on the specification that is *not* Note specification (that is, non-commentary–core and then background), as that is where the major specification defects will be found.

   Example:
   Goal [First Release]: 60% <- Marketing Director [June 6 200X]. "*Source is background, but good for credibility and SQC.*"
   Source: The Encyclopedia.

Synonyms: Notes *018; Comment *018; Remark *018.
Related Concepts: Commentary *632.
Keyed Icon: " . . . " "Double quote marks around the note."

Objective **Concept *100**

Objective is a synonym of Performance Requirement. See Performance Requirement *100.

Or **Concept *514**

'Or' is a logical operator used in qualifiers, or other appropriate specifications, to indicate alternative conditions. If any one 'Or' condition is true, then the set of conditions is true.

Example:
Stretch [If Multinational Or Government]: 99%.
Stretch [If Multinational or Government]: 99%.
Stretch [If Multinational OR Government]: 99%.

To make a statement read better, the lead capital letter may be dropped, giving 'or.' It can also be spelled all capitals, 'OR,' to emphasize that it is a Planguage logical operator and not a simple text word.

Notes:

1. Parenthesis: {Set parenthesis} and (ordinary mathematical parenthesis), may be used to limit and clarify the extent of a logical expression.

Or Better **Concept *550**

'Or Better' is an expression used within a scalar specification to explicitly emphasize that the specified level has a range of acceptable values, rather than being just a fixed, single value. In other words, 'Or Better' helps identify the specified level as the beginning of a desired range.

'Or Better' is actually implied by a scalar target specification, but it can be useful to be more explicit.

Example:
Goal [Mechanical]: 60 degrees Or Better.
Stretch: 99.90% Or Better.
Survival [Offices]: 35 degrees C Or Better.

Related Concepts: Until *551; Or Worse *549; Or Better *550; Range *552.

Or Worse **Concept *549**

'Or Worse' is an expression used within a scalar specification to explicitly emphasize that the specified level has a range of unacceptable values, rather than being just a fixed, single value. In other words, 'Or Worse' helps identify the beginning of a 'no go' range. 'Or Worse' is actually implied by a constraint specification, but it can be useful to be more explicit.

Example:
Must Avoid [EU, Next Generation Product]: 50% Or Worse.
Fail [Banking Market]: 20% Or Worse.

Related Concepts: Until *551; Or Better *550; Range *552.

Owner **Concept *102**

An owner is a person or group responsible for an object, and for authorizing any change to it.
The parameter, Owner, can be used to explicitly identify ownership.

Notes:
1. For example: a system owner, a specification owner, a standard owner or a process owner.
2. An owner is responsible for updating or changing an object, including maintaining its control information (for example, Status, Version, Quality Level and Location).
3. An owner will ensure the object adheres to any relevant standards.

Related Concepts: Stakeholder *233.

Parameter **Concept *105**

A parameter is a Planguage-defined term. Parameters are always written with at least a leading capital letter, to signal the existence of a formal definition.

Notes:
1. The master definition of most of the Planguage parameters is found in this Glossary. See also http://www.gilb.com for additional, updated and new parameters.
2. A project or specification author can declare and define tailored parameters (as part of a Project Language – a specific project specification language). These can then be reused anywhere in a specification where they are understood. They may in time be officially adopted by some local dialect of Planguage.
3. Parameters are not user-defined terms. User-defined terms are defined by a project or organization, to describe the target system, organization or project. User-defined terms are not part of the definition of a specification language.

Synonyms: Specification Language Parameter *105.

Related Concepts: Term *151; Project Language *247; User-Defined Term *530.

Past **Concept *106**

A Past parameter is used to specify historical experience, a 'benchmark'. A Past specification states a historical numeric level, on a defined Scale, under specified conditions [time, place, event] for a scalar attribute.

Notes:

1. Past values are stated to give us some interesting benchmark levels for our old system(s) and our competitors' systems.
2. Even 'current' values should be expressed using Past, because immediately they are stated, they are 'past' values. Qualifiers will make plain the currency of a specification.

Rationale: If we did not take the trouble to analyze and specify the past values then we might not set reasonable targets. Unintentionally, targets might even be specified worse than they were in the Past.

Synonyms: Past Level *106.

Related Concepts: Benchmark *007.

Keyed Icon: < "A single arrowhead, normally on a scalar arrow (<----<----O---<---->), pointing 'back' to the past. Note the '<' alone in other contexts has other meanings such as: '<' less than, '<-' (source arrow), '<--------' (tip of scalar arrow). So either it must be used in an unambiguous context or manner, or there be at least one hyphen, or [qualifier], on either side of the arrowhead to distinguish this icon."

PDSA **Concept *168**

Acronym for Plan-Do-Study-Act Cycle *168.

Percentage Impact **Concept *306**

A percentage impact is an incremental scale impact expressed as a percentage of the required improvement (the required improvement being the scalar distance between a chosen benchmark (0%) and a chosen target (100%)).

A percentage impact is part of an impact estimate and is used in Impact Estimation tables.

Synonyms: Incremental Percentage Impact *306; Percentage Incremental Impact *306; %Impact *306.

Related Concepts: Incremental Scale Impact *307; Scale Impact *403.

Percentage Uncertainty **Concept *383**

Percentage 'Uncertainty' is calculated from the scale uncertainty, baseline and target data; and stated together with the percentage impact value.

Notes:

1. Percentage Uncertainty can be used to identify risks in using specific design ideas: the numeric 'best case' and 'worst case' deviations from the Percentage Impact estimate provides important pointers towards the level of risk involved. This can lead to one or more direct actions, if the risk level is judged too high. For example (actions):
   - to specify a design better
   - to change the design itself
   - to get more information about the design impacts
   - to write contract conditions controlling the impact expected
   - to schedule this design for early evolutionary step delivery (so we can see what it really does).

Example:
If Percentage Impact = 30% and Percentage Uncertainty = ± 40%, the overall impact in percentage terms is usually stated as 30% ± 40%. In other words, Percentage Impact is assessed to vary in practice anywhere between −10% and 70%.
Dual System -> Reliability: 30 ± 40% <- Company Experience ranges from –10% to 70%.

Performance **Concept *434**

System performance is an attribute set that describes measurably 'how good' the system is at delivering *effectiveness* to its stakeholders. Each individual performance attribute level has a different 'effectiveness,' for different stakeholders and, consequently, different value or utility.
Within Planguage, performance attributes are scalar and are of three types:

- Quality: 'how well'
- Resource Saving: 'how much saved'
- Workload Capacity: 'how much'

Other possibilities exist for defining performance. For example:

"Performance. A quantitative measure characterizing a physical or functional attribute relating to the execution of a mission or function. Performance attributes include quantity (how many or how much), quality (how well), coverage (how much area, how far), timeliness (how responsive, how frequent), and readiness (availability, mission readiness). Performance is an attribute for all system people, products, and processes including those for development, production, verification, deployment, operations, support, training, and disposal. Thus, supportability parameters, manufacturing process variability, reliability, and so forth, are all performance measures."
Source: USA MIL-STD 499B Draft 1992.

Notes:

1. The system engineer or system stakeholder can select, define, invent, tailor or develop any number of useful or interesting performance measures, to serve the purposes of their current task, or systems engineering process.

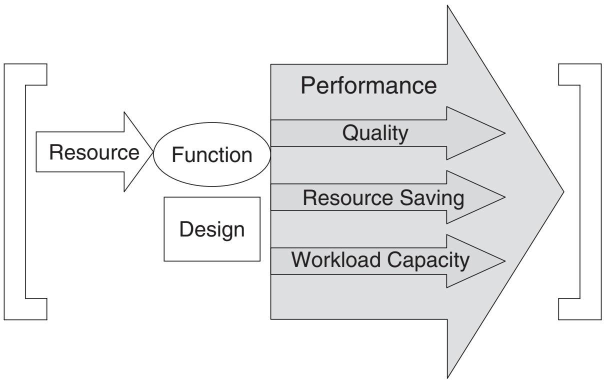


Figure G12
Performance characteristics are classified into three major types within Planguage. This is an arbitrary, but useful distinction. See also the diagram in Quality *125.

2. Performance is intended to cover absolutely all performance measures. It is not limited to the narrower conventional set of performance measures (for example, throughput speed), but explicitly includes the *qualitative* measures of performance, which are so weakly represented and too rarely quantified in conventional thinking.
3. Performance is the most general sense of how well a function is done. Performance includes:
   - quality characteristics (such as availability, usability, integrity, adaptability and portability), and
   - resource saving characteristics (such as cost reduction and reduced elapse times), and
   - work-capacity characteristics (such as storage capacity, maximum number of registered users and transaction execution speed).

Related Concepts: Quality *125; Resource Saving *429; Workload Capacity *459; Performance Requirement (Objective) *100; Performance Target (goal) *439; Performance Constraint *438; Benchmark *007; Target *489; Constraint *218.

Keyed Icon: O---> or O+ "Compare with Keyed Icon [Resource *199]: --->O or –O and Keyed Icon [System *145 [Not Design]]: {Resource, Function, Performance, Condition}: [--->O--->] or [–O+]."

Performance Constraint Concept *438

A performance constraint specifies some upper and lower limits for an elementary scalar performance attribute. These limits are either levels at which failure of some kind will be experienced, or levels at which the survival of the entire system is threatened.

Fail and Survival parameters are used to specify performance constraints.

Notes:

1. Stakeholders impose constraints. These stakeholders and their motivation should be explicitly documented together with the constraint specification (for example using Authority, Source, Rationale or Stakeholder parameters).

   Example:
   Speed:
   Scale: Time in seconds <to react>.
   Survival [Public Places]: 10 seconds maximum.
   Authority: Public Safety Law.
   Fail [All Uses of our Product]: 5 seconds.
   Authority: Our Quality Director. Rationale: Time to react to alarm light.
   Goal [Public Places]: 4 seconds. <- Project Manager.
2. Performance *constraint* (Survival and Fail) levels usually lie 'outside' the performance target (Goal, Stretch, Wish) levels.
3. Why an upper constraint limit for a performance? There are many reasons, which include:
   - To avoid 'arms race escalation'
   - To avoid unnecessary costs, which would not be valued by a market
   - To avoid costing yourself out of a market
   - To avoid unnecessary cost – contract income is constant when you reach such a limit
   - To avoid 'gold plating' and over-engineering
   - To avoid becoming a monopoly and provoking legal reaction
   - To avoid showing your hand to the opposition.

Related Concepts: Performance *434; Performance Requirement (Objective) *100; Performance Target (goal) *439; Constraint *218; Fail *098; Survival *440.

Performance Requirement — Concept *100

A performance requirement (objective) specifies the stakeholder requirements for 'how well' a system should perform.

A performance requirement can be complex or elementary. It is a scalar concept, and at elementary level is defined quantitatively by a set of performance targets and performance constraints.

Typical examples of performance requirements include 'Usability,' 'Reliability' and 'Customer Satisfaction.'

Performance Requirements are limited to consideration of the *performance* effectiveness of a system, without regard to the efficiency of it. That is, a performance requirement describes some aspect of the required performance; it does not describe the costs (the resources needed) to get the performance.

Notes:

1. The distinction between use of the terms 'objectives,' 'quality requirements' or 'performance requirements' is often simply dependent on the culture using them. Engineers are more likely to speak about 'quality requirements' for a system or product. Managers (of people and organizations) are more likely to think in terms of business/technical 'objectives.'
2. A performance requirement is a potentially complex, detailed specification. It can consist of a whole hierarchy of performance attributes.
3. For each *elementary* performance attribute (distinguished by having only one Scale), there can be *many* performance targets and/or performance constraints.

   (Note: this does not mean that the concept behind an elementary performance requirement is not 'really' somehow complex, and that

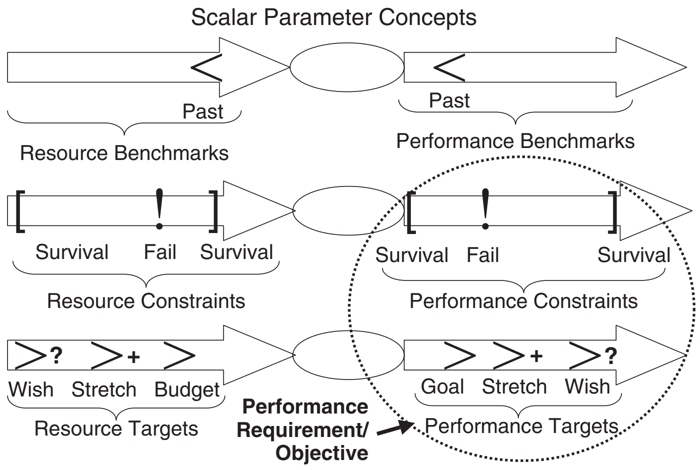


Figure G13
A performance requirement (objective) is specified as a set of performance targets and performance constraints.

it is not *capable* of further sub-setting. It is simply that for the given system, further sub-concepts are not considered to be of interest or use at the current time. So 'complex' means 'complex in terms of whether we have *decided to* decompose the concept in the current specification,' not complex in terms of some constant reality.)

Example:

We can generally speak about a performance requirement, 'Reliability' for a defined system. Reliability may well be specified as elementary (having only one scale of measure). There can be several targets and constraints specified. Here below, is an elementary Reliability performance requirement with a Fail level and two Goal level specifications. Qualifiers distinguish the Goal level specifications from each other.

Reliability: "A Performance Requirement or Objective"
Scale: MTBF.
Fail: 30,000 Hours. "*Constraint 1*"
Goal [1st Release]: 40,000 Hours. "*Goal 1*"
Goal [2nd Release]: 50,000 Hours. "*Goal 2*"

Of course, the 'Reliability' performance requirement could instead be a complex objective (that is, composed of one or more sub-objectives (elementary or complex)). For example, we might be interested in two Scales: 'Mean Time Between Failure (MTBF)' and 'Number of Repeat Occurrences of Faults.' We would specify both of them as sub-requirements of the 'Reliability' requirement.

Example:

Reliability: {Failure Rate, Repeat Failures}.

4. Additional supporting information can be present in benchmark parameters (Past, Record, Trend).

Example:

Stretch [Main Markets, Within the Decade]: 99.998%.
A Performance Target.

Example:

Security [Corporate Webservers]: Elementary Performance Requirement.
Version: 3.1. Owner: Tom. Sponsor: Simon.
Scale: Annual Frequency of defined [Type of Penetration] using defined [Type of Threat] used by defined [Type of Perpetrator].
Meter [Acceptance]: At least 300 representative cases of [Type of Threat] <- Contract 2.3.5.
================= Performance Targets =================
Goal [Security Type 1, Next Year]: <10 <- Official Project Steering Committee Agreement.
Stretch [Security Type 1, Next Year]: 0 <- Technical Director's Challenge.
================ Performance Constraint ================
Survival [Security Type 1, Next Year and On]: 60 <- CEO Public Promise of Improvement.
===================== Benchmark =====================
Past [Security Type 1, Last Year]: 66 <- Annual Executive Security Report [Page 55].
===================== Definitions =====================
Security Type 1: Defined As: Type of Penetration = Access, Type of Threat = Remote Terminal, Type of Perpetrator = Hacker].
An Elementary Performance Requirement.

Performance Requirement. "The extent to which a mission or function must be executed, generally measured in terms of quantity, quality, coverage, timeliness, or readiness."

Source: US MIL-STD 499B 1994.

Synonyms: Objective *100.
Related Concepts: Requirement *026; Performance Target (goal) *439; Performance Constraint *438.

Performance Target **Concept *439**

A performance target (goal) is a stakeholder-valued, numeric level of system performance. There are three types of performance target:

- a committed planned level (Goal),
- an uncommitted motivating level (Stretch), and
- an uncommitted valued level (Wish).

Notes:

1. The target parameters {Goal, Stretch, Wish} are used to express performance targets.

   Example:
   Goal [Main Asian Markets, Next Quarter]: 60,000 hours.
   A Performance Target.
2. A performance target is a *single* required level of performance (such as a Goal specification).

   Example:
   Goal [USA, Next Release]: 99.50%.
3. In contrast, a performance *requirement* includes both performance *targets* and performance *constraints.*

Synonyms: goal *439 (with a small 'g' to distinguish it from the parameter, 'Goal').
Related Concepts: Performance *434; Performance Requirement (Objective) *100; Performance Constraint *438; Target *048; Goal *109; Stretch *404; Wish *244.

Performance to Cost Ratio **Concept *010**

For a design idea (or set of design ideas or an Evo step), the ratio of the performance improvements to the cost of the resources needed to implement them.
For a selected set of requirements:
Performance to Cost Ratio = Sum of Performance/Sum of Costs
Rationale: Such ratios allow comparison of different design ideas to determine which is the most cost efficient. Popularly (USA), "Bang for the buck".
Notes:

1. Provided Sum of Costs is used, costs are any idea of resources, and are not limited to financial costs. This is possible because it is the *percentage* costs that are being summed. Any useful set of cost attributes may be used.
2. An alternative way to calculate the Performance to Cost Ratio is to use the sum of the absolute financial costs, rather than Sum of Costs. This can be a simpler solution as it avoids some arithmetic. It also gives the actual financial costs more prominence.
3. Keep in mind the essential distinctions between achieving the performance requirements, the consequent value to given stakeholders under

given circumstances of reaching those target levels, and the actual benefits that the stakeholders obtain. A 'value to cost' ratio calculation or a 'benefit to cost' ratio calculation, while more demanding, might give a more realistic evaluation.

A performance to cost ratio is a simpler measure: the amount of requirements satisfied to the cost incurred. It could be considered as 'how far a project is going towards meeting all its goals for what level of cost'.

Related Concepts: Sum of Performance *008; Sum of Costs *128; Value *269; Value to Cost Ratio *635; Benefit *009.

Place **Concept *107**

'Place' defines '*where*'. It relates to any notion of place, such as geographic location, stakeholder role, customer market or system component.

Place is used both as a parameter and in a qualifier condition.

Example:

Place [Person Type]: {Buddhist, Korean, Teenager}.

'Place' used as a parameter.

Goal [Place = {USA, CAN, MEX}, Date = Ten Years Time, Software Sub-system]: 60% <- North America Marketing Plan.

'Place' defined as a set of acronyms for countries.

Notes:

1. Place refers to the set of possible scope dimensions that are neither temporal, nor event dependent.
2. Place refers to notions such as:
   - geography (for example, a city, a country)
   - organizational (for example, IBM, Nokia, US Government, UK MoD)
   - types of people, including groups (for example, novice, teenager, pensioners)
   - system components (for example, hardware, radio transmitter, software, database, user terminals, chips)
   - role (manager, operator, trainee).
3. There can be any useful number of place dimensions in a single qualifier expression.

Synonyms: Space *107.

Related Concepts: Time *153; Event *062; Qualifier *124; Scope *419.

Plan-Do-Study-Act Cycle **Concept *168**

The Plan-Do-Study-Act Cycle (PDSA Cycle) is a process cycle. It is a method for changing a defined work process to reflect measurably better process. The concept was developed by Walter Shewhart and taught by W. Edwards Deming (Delavigne and Robertson 1994; Deming 1986; 1993).

Notes:

1. The PDSA Cycle involves the following:
   - Decide on an improvement and how to accomplish it. ('Plan')
   - Carry out the improvement as planned ('Do')
   - Gather data about costs and resulting improvements and side effects, analyze the data and decide what it means ('Study', sometimes called 'Check')
   - Adopt the change as is, or try again in a better way ('Act')

The ‘Plan’ phase involves specifying a theory or hypothesis, including a procedure for testing it. It sets out the order and timing of *tasks* and *events* that need to occur to achieve the ‘Do’ phase. In addition, ‘Plan’ specifies the entry and exit conditions for the ‘Do’ phase, including the criteria for terminating it early (prior to required results being achieved) and the testing procedures. It also plans the resources required. In Evo, ‘Plan’ is the planning for the delivery of the step that has just been selected.

The ‘Do’ phase involves implementing (Do . . . ing) what you have planned, that is, carrying out the ‘experiment’ to see how your plan measures up to reality. In Evo, ‘Do’ is the step implementation.

The ‘Study’ phase observes and gathers data concerning the results of the ‘Do’ phase. It is basically concerned with ‘What happened?’ It can involve highly varied analysis activities, depending on the activity being controlled, such as obtaining feedback data, carrying out specification quality control (SQC) and testing. It is a preparation for drawing conclusions and taking action in the ‘Act’ phase. In Evo, ‘Study’ is the phase where we analyze all the feedback from the last step, in relation to requirements and external impulses (like change of market or law).

The ‘Act’ phase decides on what course of action should be taken based on the information supplied by the ‘Study’ phase. It is to standardize the process at a new level, or to draw new conclusions about our original theory or to determine and select new theories (to design or modify processes). In Evo, ‘Act’ means reviewing the Evo plan, determining the gap priorities, finding alternative steps and deciding on the next step from the various alternatives. If the results of the last step are not satisfactory, a course of action to correct it might be decided upon.

2. A PDSA Cycle is enabled by standardizing and stabilizing its target process (meaning: ‘wherever you carry out the improvement’), so that the effects of any process changes can be credibly observed.

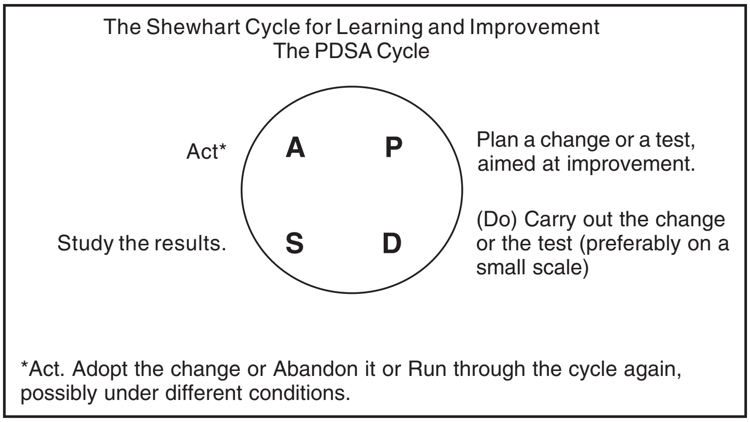

The Shewhart Cycle for Learning and Improvement
The PDSA Cycle
Act*
A
P
Plan a change or a test, aimed at improvement.
Study the results.
S
D
(Do) Carry out the change or the test (preferably on a small scale)
*Act. Adopt the change or Abandon it or Run through the cycle again, possibly under different conditions.

Figure G14
Reproduction from a letter to Tom Gilb from W. Edwards Deming, May 18, 1991.

Acronyms: PDSA Cycle *168; PDSA *168.
Related Concepts: Plan [PDSA] *169; Do [PDSA] *170; Study [PDSA] *171; Act [PDSA] *172; Evolutionary Project Management *355; Statistical Process Control (SPC) *466.
Historical Note: The Study phase has sometimes been called 'Check'. For example, Deming used Plan-Do-Check-Act (PDCA) in his book, Out of the Crisis (Deming 1986). However, in a letter to me (Tom Gilb) in May 1991, he said he strongly preferred the initial Walter Shewhart usage, PDSA, due to the word 'check' having other interpretations, such as 'stop.'

Planguage © Tom Gilb **Concept *030**

Planguage is a specification language and a set of related methods for systems engineering.
Notes:
1. Planguage specifically supports all aspects of systems engineering including requirement specification, design specification, design impact analysis, specification quality control and evolutionary project management.
2. Planguage can, however, be used much more generally; it has been used successfully to plan such diverse things as family holidays and multinational charity aid project plans.
   Planguage is designed to express ideas about any requirements, designs and plans.
3. Planguage is intended for use throughout a project lifecycle: for planning, problem-solving, specification quality control and result delivery to stakeholders.
4. Planguage has been developed by Tom Gilb, and is defined in this book. There has been lots of feedback from clients and professional friends. Its general content is described more fully in both the Introduction and Chapter 1 of this book.
5. The purpose of the copyright is to avoid being prevented later from using this term by others. Permission to use the term freely is granted by Tom Gilb when © is acknowledged.
6. If any reader finds the term 'Planguage' too 'cute,' they may use the more directly descriptive 'Planning Language.' I (Tom Gilb) often refer to 'Planning Language' before I introduce the term 'Planguage.'

Synonyms: Planning Language *030.

Priority **Concept *112**

A 'priority' is the determination of a relative claim on limited resources. Priority is the relative right of a competing requirement to the budgeted resources. If resources were unlimited, there would be no need to prioritize things. You *could* 'have it all.'
An explicit Priority parameter can be used to specify any direct priority relationships.
Notes:
1. The specified, qualified, stakeholder requirements (the targets and constraints stated in the requirements) provide 'natural' (requirement related), and dynamically computable, priority information. The gaps remaining until the goals are met, and budgets used up, can be measured and computed. In general, the largest gap to a performance target will have the highest priority, and the largest gap to using up a budgeted resource will indicate the safest resource opportunity.

However, to determine your priorities in specific cases, the degrees of risk and uncertainties, and/or knowledge of the effort (the resources) needed to close the individual gaps to meet each of the function and performance targets, and the likely resulting stakeholder values on delivery, all need to be taken into account (see below for a more complete list).

2. Priority is not a constant. It cannot and should not be determined and specified in the form of static priority weighting numbers (for example, '25%,' or third on the priority list) or words (for example, 'High'). Current priority depends on how well satisfied the competing performance targets are and how 'used up' the budgeted resources are at any given time. Current priority also depends on the more fundamental changes that can occur in requirements themselves, as stakeholders modify their requirements, and as the external business environment alters – to demand requirement changes.

   Example:
   Consider your priorities for food and air. If you are hungry, then you give priority to eating. However, as soon as air is in short supply, your priorities change. Your body gives priority to breathing.
   Your body knows your food and air requirements, both targets and constraints. Your body knows the current supply levels of these resources. When changes in the body resource levels dictate change in our priorities, this knowledge triggers the body to appropriate action. The body, as a system, acts in order to ensure our comfort and survival.
   Priority is dynamic.

3. Priority is decided by a wide variety of factors, which include but are not limited to:
   - qualifier conditions (factors such as timescales and location)
   - stakeholder authority
   - stakeholder influence
   - consequences of failure (not meeting Fail constraint levels)
   - consequences of catastrophe (not meeting Survival constraint levels)
   - previous experience of meeting similar requirements (including no experience!)
   - complexity of meeting the requirements
   - consequences of success (primarily meeting the *performance* targets, the Goal levels: the system improvements delivered, and the benefits likely to be experienced)
   - resource availability (or maybe more significantly, resource unavailability)
   - dependencies.

   Within Planguage, the relative priority of a requirement depends on a combination of the elements of the specification. These elements include target levels (for example, Goal and Budget), constraint levels (for example, Fail and Survival), its qualifier conditions [time, place, event], and its authority level.

   If you consider only the different requirement level parameters as a class: first priority is satisfaction of all the constraints; the Survival levels, then the Fail levels. Then next priority becomes satisfaction of targets; the Goal levels, after that any Stretch goals are considered, and finally perhaps some Wish levels.

However, you have also to consider the qualifier conditions as well, for all these levels, as qualifiers bring into play additional factors, like the timescales for the requirements to be met, and the events under which the requirements actually exist.

No priority exists until the qualifier conditions [time, place, event] warn us of potentially unfulfilled requirements. Targets and constraints are not finally effective until their qualifier conditions are true, but the designer, the architect and the project manager have to prioritize their contribution in *advance* of deadlines, and other conditions, becoming 'true.'

4. Stakeholders' needs will ultimately decide the relative priorities. There will be trade-offs to consider when there are conflicts between requirements. System designers should evaluate priorities and then present the results for confirmation, selection or conflict resolution to the stakeholders themselves. Stakeholders, as a result of seeing the cost and feasibility of design options, may then choose to change some of their priority specifications.
5. The Priority parameter can be used to help people to more directly understand the priorities (or to confirm the derived priorities with stakeholders). Alternatively, it can be used to specify priorities that differ from what would otherwise be expected or evaluated.
6. Rationale and Source parameters should ideally support Priority parameter specifications.

Example:
Usability:
Scale: <Speed of mastering> defined [Tasks] by defined [Staff Type].
Fail [USA, Task = Query Handling, Staff Type = Customer Service]: 35 hours.
Fail [Europe, Task = Query Handling, Staff Type = Junior Management]: 25 hours.
Goal [Australia, Task = Query Handling, Staff Type = Customer Service]: 30 hours.
Priority [Usability]: Australia Before Europe Before USA <- Marketing Plan 6.5.
Rationale: Past bad experiences with current system.
Source: Technical Director.
Without the explicit 'Priority' statement we would normally prioritize the Fail levels. However, the Priority specification means we should use scarce resources on the Australia Goal before we use them on the two Fail levels. We should then use resource on Europe before the USA.

Related Concepts: Level *337; Risk *309; Gap *359; Dependency *189.

Procedure — Concept *115

A procedure is a repeatable description to instruct people as to the best-known practice, or recommended way, to carry out the task of a defined process. A procedure is part of a process description.

"No matter how many theorists have advocated a procedure, if the procedure has been given a thorough trial and then abandoned, there is a strong presumption that it is unsound."
<-(Mintzberg 1994 Page 135, quoting R. N. Anthony, 1965, Page166, in Planning and Control Systems, Graduate School of Business Administration, Harvard University).

Notes:
1. The procedure description can be kept and presented in various forms including {forms, guidelines, diagrams, video and audio; even unwritten, but understood procedures}.
2. A procedure describes a task that there is benefit in standardizing (usually, because it is carried out so often, to ensure best practice and to prevent error).

Related Concepts: Task *149; Process *113.

Process — Concept *113

A process is a work activity consisting of:
- an entry process, which examines entry conditions
- a task process, which follows a procedure defining the task. There might also be an associated verification process, such as test or quality control
- an exit process, which examines exit conditions.

Processes transform inputs to outputs, using resources, and display their own performance and resource (cost) characteristics.

Notes:
1. A process is a set of actions, carried out by people, nature or machines (agents), which can be defined by inputs, outputs, a sequenced set of actions and its performance, and resource attributes.
2. Processes can only be fully understood by including information about the agent who executes the process (their work environment, competence, experience, training).

Process: "A system of activities that use resources to transform inputs to outputs."

Source: ISO 9000, 2000.

Synonyms: Work Process *113.
Drawn Icon: A rectangle with up arrow on left side.
Related Concepts: Task *149.

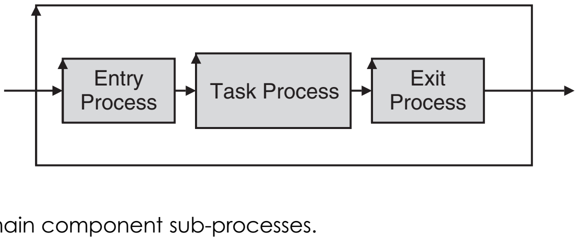


Figure G15
A process with its three main component sub-processes.

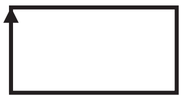


Figure G16
The drawn icon for Process *113.

Process Improvement **Concept *114**

Process improvement is systematic activity, which consists of:

- determining the root systemic causes of human work process problems, then
- changing defined and stable work processes, with the intention of eliminating, or reducing, the human tendency to commit errors, and therefore improving productivity or costs.

'Root causes' are the earliest defect causes in the chain of all causes and effects. It pays to remove the source of a problem. 'Systemic causes' are those that are built into the process, and bound to cause a problem regularly if not improved. The proof of improvement lies in the results from a changed process.

Related Concepts: Root Cause *263; Systemic *262.

Qualifier **Concept *124**

A qualifier is a defined set of conditions embedded in, or referenced by, a specification. All of its conditions must be 'true,' for that specification to apply. A qualifier defines any interesting set of *specific* time, location and event conditions (also known as qualifier conditions). These are sometimes called 'when,' 'where' and 'if' conditions.

Square brackets around the qualifier conditions are used to denote a qualifier. An alternative is to use the Qualifier parameter (which also happens to use square brackets!).

Example:

Goal [Germany, Teachers]: 65%.

Where 'Germany' and 'Teachers' are each qualifier conditions (defined elsewhere). The set of qualifier conditions, and the square brackets, form the qualifier.

Example:

Goal [German School]: 65%.

German School: Qualifier: [Country = Germany, User = Teachers].

Where 'German School' is a defined reusable qualifier. Tagging a 'Qualifier' parameter or statement allows us to simplify a large set of qualifier conditions, and perhaps to describe or summarize them at the same time. 'German School: [Country = Germany, User = Teachers].' would be the logical equivalent to the 'Qualifier:' statement above. You can tag a qualifier directly.

Notes:

1. Qualifiers can be present in any specification (for example, a system attribute specification, a requirement specification, a design idea specification or an Evo step specification). Most Planguage parameter specifications are subject to qualifiers (either explicit, implied or inherited).
2. Any number of qualifier *conditions* can apply to a given specification, expression or statement. There can be multiple instances of any one class of qualifier conditions. There is no sequence requirement for the conditions.
3. Qualifiers are always specified as sets within square brackets, '[ ]', even when a 'Qualifier' parameter is used.
4. Qualifiers have to be evaluated to see if they are 'true' in any given instance. In the case of evaluating an Evo step, this may be done in real time.

5. Qualifiers have the effect of 'dividing up' a system into separate, maybe overlapping, system dimensions or 'views.' This has many uses. One use is to divide up a system into smaller distinct areas for delivery of Evo steps.
6. A qualifier is a substantial contribution to understanding the *priority* of a specification.

   Example:
   Project Manager Attendance:
   Goal [By = Next Year, Market = London, Activity = Customer Meetings, Event Condition = If Sale Agreed]: 90%.
   The qualifier conditions specify when, where, during which activity, and after which event – the Goal has validity – and thus has priority over any specification that is not yet *valid.*
   MOP:
   Scale: % Uptime.
   Goal [QQ]: 99.5%.
   Stretch [QQ]: 99.9%
   QQ: Qualifier [By End of Year, Home Market, Consumer Goods, If Fierce Competition on Price].
   Authority [MOP]: Product Planning.
   The MOP requirement has two distinct priority mechanisms. The MOP Goal statement has priority over a corresponding ('has same qualifier') Stretch statement. Secondly, the QQ tagged qualifier has a number of qualifier conditions that must all be true for either of the target levels to be in force. The Authority statement gives additional prioritization information for MOP, in relation to other requirements with different powers behind them.
7. A qualifier defines the set of conditions, which together enable, or 'activate', a related statement. The potential qualifier conditions can be *roughly* classified as:
   - **time** conditions: '*when*': Dates, deadlines, relative times to events, weekdays, hours (time spans or precise hour)
   - **place** conditions: any notion of '*where*': Geographical location; type of person, group or role (like trainee, teenager, teacher); system component (like module, program, laptop screen, software, contracts, standards)
   - **event** conditions: any notions of '*if* ': Any occurrence conditions such as:
     - activity commenced or terminated (like project started, policy issued)
     - activity in progress (like testing being carried out, parliament in session, voting in progress)
     - specific indicator set (like red light is on)
     - specific status attained (like Approved, Checked).
   - We can optionally preface event conditions with the logical 'If' parameter in order to emphasize that we must analyze the status of the event to determine if the qualifier condition is 'true'.

   Example:
   Fail [Europe, Year = After Ten Years, Peace]: 60% ± 20% <- Annual Plan Section 6.4.5.
   The three qualifier conditions must all be 'true' for the '60%' requirement level to be a valid requirement. 'Peace' is an example of an event condition. Europe is a place condition.

8. The implication of the qualifier definition, that all qualifier conditions must be 'true' to have effect, is that in a qualifier like [A, B, C], 'A *and* B *and* C' must be valid for the qualifier to 'enable' its related statement.
9. Here are some notes on how the concepts of qualifier, qualifier condition and scale qualifier relate to each other.
   Qualifier = [qualifier condition 1, qualifier condition 2, . . . qualifier condition n]
   Scale Qualifier = [sets up a need for *one* qualifier condition to be specified on use of the Scale]
   Scale Variable = a value for a qualifier condition satisfying a scale qualifier in a statement referring to the defined Scale..
   Scale: . . . . [scale qualifier 1] . . . . . . [scale qualifier 2] . . . . . . [scale qualifier n].
   Parameter [Qualifier]: assigned numeric value.
   Remember a Qualifier is a set of qualifier conditions:
   Parameter [{qualifier conditions}]: assigned numeric value.
   Qualifier = [qualifier condition 1 = scale qualifier 1 = scale variable 1,
   qualifier condition 2 = scale qualifier 2 = scale variable 2,
   . . . ,
   qualifier condition n = scale qualifier n = scale variable n,
   other qualifier conditions as needed not related to the Scale].
   Scale: . . . . . . [scale qualifier n: Default = scale variable n].

Related Concepts: View *484; Time *153; Place *107; Event *062; Condition *024; Scale Qualifier *381; Indicator*195; Status *174.
Keyed Icon: [ ] "Square brackets around any set of qualifiers."

Quality — Concept *125

A quality is a scalar attribute reflecting 'how well' a system functions.

Example:
Quality: Includes: {Availability, Usability, Integrity, Adaptability, and many others}.

Notes:
1. A quality is a system performance attribute. All systems have a large number of quality attributes in practice. In a given situation, only the relevant quality attributes will be specified: these are the qualities specifically valued by the stakeholders.
2. All qualities can be described numerically using a defined Scale, or set of Scales. Existing quality levels can be specified as benchmarks, and needed future quality levels can be specified as targets and constraints.

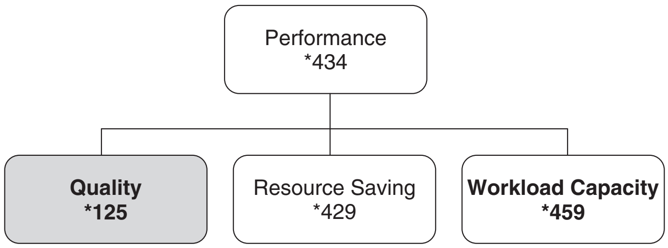


Figure G17
Quality viewed in the context of Performance.

3. Quality is distinct from the other performance attributes: Work Capacity and Resource Saving.
4. **Quality** is characterized by these traits:
   - Quality describes *'how well'* a function is done
   - Quality is *valued* to *some* degree by *some* stakeholders of the system
   - *More* quality is generally *valued* by stakeholders; especially if the increase is free, or at lower cost, than the value of the increase
   - Quality attributes can be *articulated* independently of the specific means (designs) used for reaching a specific quality level – even though all quality levels *depend* on the particular designs used to achieve them
   - A specific quality can be a described in terms of a *complex* concept, consisting of multiple complex and/or elementary quality concepts
   - Quality is *variable* (along a definable scale of measure: as are all scalar attributes)
   - Quality levels are capable of being specified *quantitatively* (as are all scalar attributes)
   - Quality levels can be *measured* in practice
   - Quality levels can be traded off to some degree; with other system attributes valued more by stakeholders
   - Quality can never be perfect in the real world
   - There are some levels of a specific quality that may be outside the state of the art at a defined time and under defined circumstances
   - When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity

Related Concepts: Performance *434; Resource Saving *429; Workload Capacity *459; Quality Requirement *453.

Quality Level — Concept *360

The quality level of a specification is a measure of its conformance to any specified relevant standards.

The Quality Level parameter is used to specify the estimated defect density for a specification: in other words, the number of estimated remaining major defects/(logical) page.

Synonyms: Major Defect Density *360; Specification Quality Level *360.

Related Concepts: Status *174; Remaining Major Defects/Page *060; Specification Defect *043; Major Defect *091; Logical Page *103.

Quantify, To — Concept *385

To quantify is to specify numerically.

Notes:

1. To articulate a variable attribute using a defined scale of measure and specifying one or more numeric levels on the Scale. The resulting specification is 'quantified.'
2. In particular, we need to be clear that 'to quantify' is not identical to the concept of 'to measure.' Measuring is the act of determining where we are, on a defined scale of measure, in practice by using a Meter.
   Quantification is a type of specification that is a prerequisite for other processes, such as Estimation, Measurement, Testing, and Feedback Control.

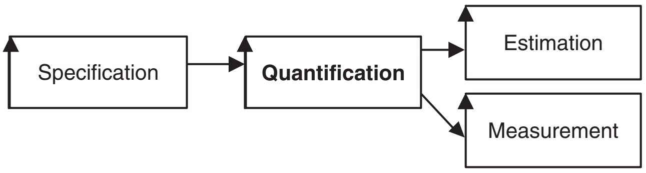


Figure G18

3. Quantification must not be confused with estimation, either. You can quantify without estimation and without measurement! Estimation is to determine a particular (qualifier specified) past or future number, on a defined scale of measure.
4. In Planguage, quantification begins with the definition of a Scale. Quantification continues, and takes on more specific meaning by using benchmarks, targets, assumptions and the many other specification parameters that relate to the defined scale of measure.

Dictionary Definition of 'Quantify':

"1. To determine or express the quantity of; indicate the extent of; measure
2. To express in quantitative terms, or as a numerical equivalent logic to make the quantity or extension of a term or symbol clear and explicit by the use of a quantifier, as all, none, or some."
<- Webster's New World^{TM} College Dictionary *(Third Edition).*

The Planguage definition of quantify (identical to variant 2 above) does not admit to the term 'measure' which is contained in variant 1 of this definition, and in common use of the word. In Planguage, 'measurement' (as in variant 1) is a quite distinct activity, based on the quantification (but not identical to it).
Related Concepts: Scale *132; Meter *093; Measure, To *386; Estimate, To *059.

Range — Concept *552

A range is the extent between and including two defined numeric levels on a scale of measure.

Notes:

1. Range captures a snapshot of some common scalar attribute ranges (using the target and constraint levels). Some examples of different range classifications include:
   - Success Range – project or product success
   - Acceptable Range – not success, but not failure
   - Failure Range – some degree of problems or failure
   - Catastrophe Range – forget it! No use and no hope!
2. A range is usually *implied* by specification of the relevant levels. For example, a performance success range goes from the applicable Goal level in the direction of 'better' forever, unless bounded by an *upper* Survival level.
3. Range, like a level, is interpreted with regard to any specified qualifier conditions. In other words if a scalar level qualifier, for example for a Goal level, is not valid, the range implied by that Goal level is not valid either.
4. Different stakeholders can specify levels with implied ranges that overlap with each other. 'One man's meat is another man's poison,' as the

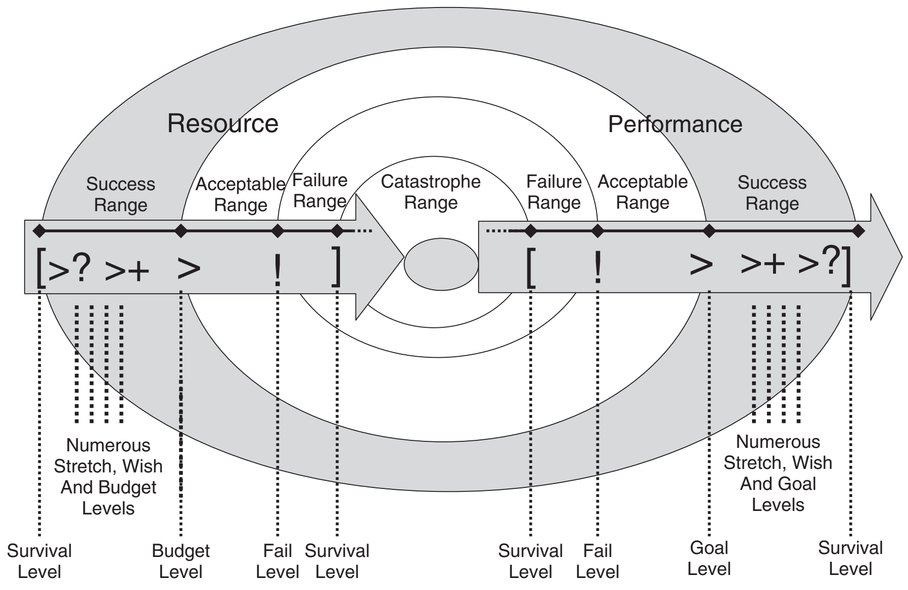


Figure G19
The 'doughnut' diagram indicating different range concepts.

saying goes. Designers need to consider the set of overlapping ranges in an attempt to maximize design efficiency. This is complex, but the problem can at least be clearly seen.

5. A range can include the idea of a set of benchmark levels over time and/or space. For example, a quality level range in a defined market for defined types of people for a defined task during annual periods.
6. A range can describe a gap between a benchmark and a target (the gap is the unfulfilled requirement).
7. A range can describe the change in attribute level as a result of carrying out an Evo step.

Related Concepts: Landing Zone *605; Gap *359; Success Range; Acceptable Range; Failure Range; Catastrophe Range.

Rationale — Concept *259

A rationale is the reasoning or principle that explains and thus seeks to justify a specification. 'Rationale' is a parameter for declaring information that justifies a specification.

Notes:

1. The information can concern the logic, the politics, the economics or whatever is of interest to declare in order to explain and justify a specification.
2. The purposes of a rationale are:
   - to answer questions that readers would ask of a specification
   - to motivate and convince readers about a specification
   - to set up information for risk analysis (is the given rationale true, and will it be later?)

Example:
PGB: Goal [UK]: 99.9% <- Annual Plan.
Authority: Board of Directors, Jan 25th.
Rationale [PGB]: Competition in UK prior to new EU Laws about competition.
Basis: Our long-range plan to be the <biggest> in all European countries.
In the example above, the PGB tag is inserted to show how to tie any Rationale statement to another specific statement or statements. This format can be used irrespective of where you specify the Rationale statement. It does not have to be just below or in the immediate vicinity. The Authority and Basis statements are implied to be related, because they are just below the PGB statement.
Note. 'Basis' is quite different from Rationale. Rationale is a set of conditions leading to a desire to make a specification. It explains how we got to that specification. Basis is a specified set of assumptions that underlie a specification. If the basis conditions are changed, then the specification may no longer be valid.

"Theirs is but to reason why: The value of recording rationale."
<-(Hooks and Farry 2000 Chapter 8)

Synonyms: Justification *259; Reason *259.
Related Concepts: Basis *006.
Historical Note: 'Rationale' is a term I found used by Synopsys, CA USA 1996.

Readership — Concept *295

The readership (or intended readership) of a specification is all the 'types of people' we intend shall read or use the specification.
Notes:
1. The 'readership' should be explicitly stated within document header information or other appropriate place. Experience has shown that allowing people to guess what the intended readership is will lead directly to not satisfying some important types of readers.

Rationale:
- It helps authors decide on document content and what terminology to use. For example, which abbreviations and terms might be understood or misunderstood.
- It also helps checkers decide if the writer has communicated clearly and unambiguously with their intended audience.

Example:
Readership: {All Employees, Customers, Suppliers, Contractors, Auditors}.
Synonyms: Intended Readership *295.

Record — Concept *127

A Record parameter is used to inform us about an interesting extreme of achievement. A Record specification states a benchmark level on a defined Scale under specified conditions [time, place, event] for a scalar attribute that represents an impressively good level, or state-of-the-art.

Example:
Usability:
Ambition: More user-friendly than <competitors' products>.
Scale: Average Learning Time for Slowest Learners in User Population.
Trend [Main Competitor, Next Year]: 1 minute.
Record [Last Year, UK]: 2 minutes <- Industry Statistics [Last Year].
Fail [New Product, Next Year]: 50 seconds.
Goal [New Product, Anytime]: 20 seconds.
It is good practice to indicate the source (<-) of Record information (as with all scalar level specifications).

Rationale: Record is usually specified to demonstrate that such a level is technically possible under certain specified conditions, and to challenge us to strive to avoid, approach, meet or beat that level, as appropriate.
Levels approaching state-of-the-art are useful to specify, because they tend to be costly and high risk.
Synonyms: Record Level *127.
Related Concepts: Benchmark *007.
Keyed Icon: <<
"The arrow points towards the 'past' as in the Past benchmark, '<' but doubled for emphasis to show that this is an extreme benchmark. Usually used in the context of a scalar arrow (<---<<---O---<<--->)."

Relationship — Concept *142

A relationship is a connection between objects.
Related Concepts: Object *099; Interface *194; Hierarchical *083; High-Level *082; Supra *264; Downstream *052; Upstream *291; Set *133; Subset *222; Is Part of *621; Consists Of *616; Includes *391; Kin *353; Sibling *265; Kid *266; Parent *267; Dependency *189: Synonym is Depends On; Impacts *334; Is Impacted By *412; Is Supported By *414; Supports *415.

Requirement — Concept *026

A requirement is a stakeholder-desired, or needed, target or constraint. Within Planguage, requirements specifically consist of vision, function requirements, performance requirements, resource requirements, condition constraints and design constraints.
Given below are some IEEE definitions:

> *Requirement:* "a condition or capability needed by a user to solve a problem or achieve an objective."
>
> *<- IEEE 610.12-1990.*

Example of an IEEE definition of 'requirement': the term 'user' is probably not broad enough to capture the scope of all stakeholders. Another danger of this definition is that it inadvertently includes all designs, and so does not successfully made a sufficiently clear distinction between requirement and design.

> *Requirements:* "Statements, which identify the essential needs for a system in order for it to have value and utility. Requirements may be derived or based upon interpretation of stated requirements to assist in providing a common understanding of the desired operational characteristics of a system."
>
> *<- IEEE P1220. IEEE Standard for Systems Engineering, Preliminary, 1993 in (SEI CMMI 1995).*

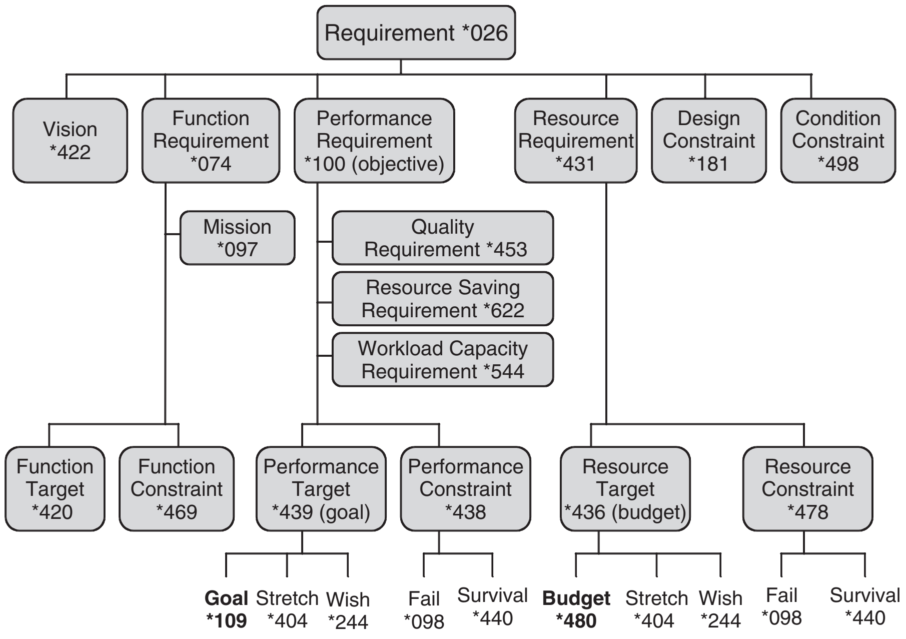

Requirement *026
Vision *422
Function Requirement *074
Performance Requirement *100 (objective)
Resource Requirement *431
Design Constraint *181
Condition Constraint *498
Mission *097
Quality Requirement *453
Resource Saving Requirement *622
Workload Capacity Requirement *544
Function Target *420
Function Constraint *469
Performance Target *439 (goal)
Performance Constraint *438
Resource Target *436 (budget)
Resource Constraint *478
Goal *109
Stretch *404
Wish *244
Fail *098
Survival *440
Budget *480
Stretch *404
Wish *244
Fail *098
Survival *440

Figure G20
Requirement Concepts.

Example of an IEEE definition of 'requirements': better than the previous 1990 standard, as all designs are no longer 'in the picture.'

Notes:

1. Stakeholders should determine their own requirements; they certainly should be involved in discussions about the relative values, costs and priorities of their requirements.
   A systems engineer may be needed to specify the requirements in a suitable way for use by systems engineering projects. Also, there may be a need for building, aggregating and analyzing a set of project requirements across a range of disparate stakeholders.
2. Not all requirements initially specified are necessarily accepted for actual delivery: some requirements may not ultimately be feasible or economic. The key practical idea is to try to identify, and give priority to, the most critical or most profitable stakeholder requirements.
3. A requirement is an input to a design process. Requirements give information to the designer about the necessary nature of their design. A design, whether a specification or an actual implementation, can be judged (using such means as Specification Quality Control (SQC), test, Impact Estimation tables, evolutionary step feedback, or operational use) in terms of how well it satisfies the requirements.
4. Requirements, at different levels of abstraction, can be viewed as inputs to a defined level of design process. In a series of systems engineering processes, one engineer's output ('design,' 'architecture') may become another engineer's inputs or 'requirements'. The conclusion of this is that specifications, no matter what we name them, are requirements only when they are used as input to such a systems engineering process.

So, we need to explain a particular concept of 'requirement' by first specifying the systems engineering process type, or level at which they apply. For example, stakeholder value analysis, product line architecture or component engineering.

5. Requirements can have different levels of priority. Priority is conveyed in a variety of ways, for example:
   - for scalar attributes, by stating any relevant constraint levels using {Fail, Survival}
   - for binary attributes, by stating any relevant constraints using Constraint parameters
   - for scalar attributes, by setting relevant target levels using {Goal/ Budget, Stretch, Wish}
   - by specifying different qualifier conditions for [time, place, event] (Alternatively known as [when, where, if])
   - by specifying other parameters, like Authority, Dependency, Priority and Risk, which provide additional information
6. Requirements are usually assumed to be written in requirement specifications. However, many documents which have 'requirements' in their title may contain little or no *real* requirements: it is unfortunately commonplace to find a high content of design specification for unspecified or vaguely specified requirements. Frequently, the most important, high-level requirements are not stated clearly at all. You must be prepared to look for requirements in other documentation and to ask questions.
7. A design idea is *not usually* a requirement at the *same* systems development process level as the requirements it was designed to satisfy. However, once a design idea is 'fixed' at a project development process level, it becomes a 'design constraint' (which is a requirement type) for the next level. In other words, a design idea usually becomes 'valid as a requirement' at the next level of the development process, after the level it was 'designed.' It is then a valid 'requirement' for all future 'lower' levels.

Example:
Usability:
Type: Quality Requirement.
Scale: Time to learn [defined Task].
Goal [By Initial Delivery, CW-Task]: 30 minutes. "A simple requirement goal."
CW-Task: Defined As: <mastering> <frequent tasks> at the <standard workstation>.
Interface:
Type: Design Idea.
Specification: The computer terminal interface must look exactly like our old one.
Impacts: Usability. "A design to meet the Usability requirement."
Usability is a strategic requirement. Interface is a design idea that we hope will satisfy the planned level for Usability. Once adopted, Interface is handed to others in the development process, such as estimators, constructors and testers, and is then classed as a design constraint, at this lower level.

Related Concepts: Requirement Specification *508; Need *599; Problem Definition *598; Problem *270; Target *048; Constraint *218; Stakeholder *233; Objective *100.

Requirements Engineering **Concept *614**

Requirements Engineering is a requirements process carried out with an engineering level of rigor. Requirements Engineering includes all aspects of requirements gathering and maintenance, including but not limited to:

- requirements solicitation (including stakeholder analysis)
- requirements analysis
- requirements quality control
- requirements review
- requirements change management
- requirements specification (the process)
- contracting and bidding requirements
- requirements risk analysis
- requirements priority analysis.

Synonyms: Requirement Engineering *614.

Related Concepts: Requirement *026; Requirement Specification [Process] *634.

Requirement Specification [Specification] **Concept *508**

A requirement specification is a defined a set of requirements. It also includes any relevant requirement background, such as benchmarks, and also any appropriate commentary.

Note:

1. A requirement specification is the output of a requirement specification process, which is a subset of a requirement engineering process.

Template: Requirement Specification Template.

Synonyms: Requirements Specification *508.

Related Concepts: Requirements Engineering *614; Requirement Specification [Process] *634; Requirement *026; Specification *137; Commentary *632; Background *507; Core Specification *633.

Resource **Concept *199**

A resource is any potential system input. A resource is any kind of input 'fuel' necessary for building, operating or maintaining a given system. A resource is an asset or a supply that can be used to produce the functionality or performance levels of a system.

A resource can be defined using a scale of measure. A requirement for a resource can be specified by a target or a constraint level. Previous levels of resource utilization (costs) can be specified by benchmark levels, like Past.

We can distinguish between budgeted and unbudgeted resources. Resource budgets are found in our formal plans.

Notes:

1. The emphasis is on the concept of '*potential*' resource. A *potential* resource is the total resource that might *theoretically* be consumed, used, applied or produced. This is in contrast to a *level* of that resource that we *plan* to use, called a Budget (a resource target).
2. We should not plan to use more than the resource actually available at any point in time, place or circumstances. However, when one type of resource is *unavailable*, we can consider the possibility of employing *another resource* to achieve our aims; this is one kind of tradeoff. The classic example is 'time versus money.'

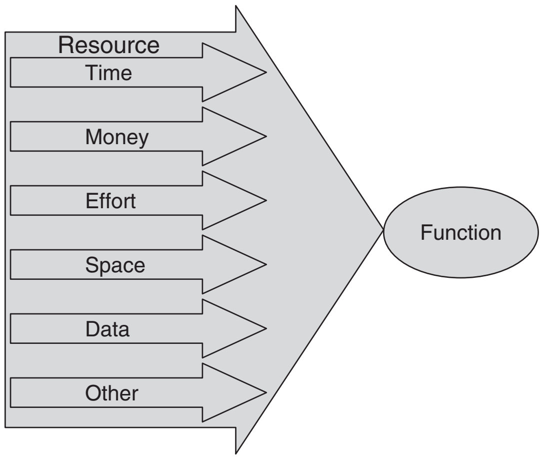


Figure G21
Arbitrary examples of some system resources.

3. Resources must be viewed with regard to the 'total potential resource available – now and later,' under the defined conditions. For example: 'in our divisional budget,' 'within the market window' or 'by the contract handover deadline.'
4. When you plan to limit the use of specific resources, you do so by setting a resource target (for example, 'Budget: €2 million.') or a resource constraint (for example, 'Survival: €2.2 million.').
   You might also specify a global or policy constraint (see example below).
   Example:
   Innovation Constraint [Division A]:
   Type: Constraint [Financial Resource].
   Scale: % of the annual research budget.
   Goal: 20%.
   Authority: Divisional Manager.
5. Resources that are input to a system differ from resource savings. Resources consumed are the costs of developing and operating a system. A resource saving is a relative reduction in consumption of a resource once a system is operational. For example, systems engineering effort as an input resource can be applied to save system user learning time (a resource saving).
6. Common usage of the term 'resource' in the United States (USA) is to mean 'people.' The Planguage definition is far wider than this.

Related Concepts: Resource Requirement *431; Resource Target *436; Resource Constraint *478; Benchmark *007; Target *489; Constraint *218; Cost *033; Resource Saving *429.

Keyed Icon: --->O "A scalar attribute arrow into a function oval. A simplified alternative is '–O'."

Resource Constraint — Concept *478

A resource constraint is a resource requirement, which specifically *restricts*, or *serves as a warning* about, the level that can be used of a resource.

A resource constraint is specified as a scalar resource attribute level. It signals the level at which *some degree of system failure* will be experienced or the level at which the entire system becomes threatened.

Two main parameters can be used to specify these constraint levels: Fail (Fail and worse might be recoverable) and Survival (worse is unrecoverable).

Notes:

1. Resource constraints are imposed or suggested by defined stakeholders. These stakeholders and their reasons should be explicitly documented with the constraint level, for example, by using Authority, Source, Rationale or Stakeholder parameters.
2. The Fail and Survival concepts are adequate for most scalar constraint purposes. However, Catastrophe is also available for use. It is a matter of taste.

Related Concepts: Constraint *218; Performance Constraint *438; Resource Target *436; Fail *098; Survival *440; Catastrophe *602; Range *552: See Failure Range and Catastrophe Range.

Example:
Memory Space:
Type: Resource.
Scale: Gigabytes of defined [Memory Component].
====================== Targets ======================
Wish [Memory Component = Online Backup]: 1,000 Gb <- Design Team.
Rationale: Improves Recovery Speed.
Stretch [Memory Component = Online Storage, US Market]: 500 Gb? <- Marketing.
Budget [Memory Component = Primary]: 100 Gb <- Initial Software Size Estimates.
=====================Constraints =====================
Fail [Memory Component = Online Storage, US Market]: 250 Gb Or Less? <- Marketing.
Rationale: Large Scale Users must have this level <- US Sales.
Survival [Memory Component = Online Storage, US Market]: 100 Gb? <- Marketing.
Rationale: Nobody would even consider our system with less <- US Sales.
Some examples of Resource Constraint specification.

Resource Requirement Concept *431

A resource requirement specifies how much of a resource should be made available for later consumption. A resource requirement is a scalar attribute with one or more resource targets and/or resource constraints specified.

Example:
Maintenance Expenditure:
Scale: Million € annually.
Budget [First Four Years Average]: 3 million €.
Fail [Any Single Operational Year]: 4 million € <- Client limit in contract §6.8.

Example:
Development Effort:
Scale: Engineering Hours applied to the contract.
Wish [Europe Release]: 10,000 hours <- Project Manager.
Survival [All Releases]: 30,000 hours <- Maximum corporate availability by deadline.
An example showing a resource target and a resource constraint specified for each resource requirement.
Related Concepts: Resource *199; Resource Target *436; Resource Constraint *478.

Resource Saving **Concept *429**

A resource saving is a performance attribute of a system. It expresses 'how much' better the system currently performs in terms of resources than it did at some previous benchmark time.
For example, a resource saving can express how much less resource in training effort or maintenance cost is needed in one system compared to another. This measure can be used for benchmarking or for setting requirements, or even for reporting on progress in design or actual measured implementation of new systems.
Synonyms: Saving *429.
Related Concepts: Performance *434; Quality *125; Workload Capacity *459; Resource Saving Requirement *622.

Resource Target **Concept *436**

A resource target is a budget. It is a scalar requirement; a resource level we aim, or might possibly aim, towards keeping within while working towards achieving the other requirements.
Three parameters are used to specify resource targets {Budget, Stretch and Wish}. A Budget level is the *primary* resource target type. A Stretch level represents a resource target that is *not* committed, but is a level for challenge and motivation. The Wish level represents a resource level that would have value to some stakeholder, but again is not committed.
Resource targets represent the reasonable, perhaps profitable, levels of cost, we must expect to pay to reach our performance and function targets within any constraints. They differ from resource *constraints*, which are the levels that signal problems, danger or lack of profitability. We do not plan and design to merely stay *within* resource constraints, but to *avoid* going anywhere near them at all!

Example:
Memory Space:
Scale: Gigabytes of defined [Memory Component].
Wish [Memory Component = Online Backup]: 1,000 Gb <- Design Team.
Rationale: Improves Recovery Speed.
Stretch [Memory Component = Online Storage, US Market]: 500 Gb? <- Marketing.
Budget [Memory Component = Primary]: 100 Gb <- Initial Software Size Estimates.
Synonyms: budget *436 (with a small 'b' to distinguish it from the parameter, 'Budget').

Related Concepts: Budget *480; Resource *199; Target *048; Wish *244; Stretch *404; Resource Constraint *478; Resource Requirement *431; Requirement *026.

Result Cycle **Concept *122**

Within Evo, a result cycle is an entire Evo step cycle aimed at delivering a result that moves towards satisfying the overall requirements.

Notes:

1. A result cycle is a cycle consisting of 'Plan-Do-Study-Act' activities.
2. It can involve any kind of system change, small or large: for example, factory production modification, software program alteration, organizational restructure, new software product development and design of new businesses.
3. A project using Evo will execute numerous result cycles. The emphasis is on 'contact with reality' and *using consequent feedback to adjust.*
4. A result cycle consists of:
   - a strategic management cycle: A strategic management cycle is concerned with controlling and monitoring the overall change process. Amongst other things, a strategic management cycle approves proposed changes against strategic objectives and adopted high-level strategies. It co-ordinates with other programs and projects. It acquires development budgets. It analyzes feedback measurements. It decides the next step.
   - an implementation cycle: Implementation means 'taking a plan, or an idea, and turning it into reality.' An implementation cycle consisting of the following sub-cycles:
     – a development cycle: This is an optional backroom cycle concerned with acquiring/purchasing and developing, any products required for the production and/or delivery cycles. For example, any new systems development would be carried out within this cycle.
     – a production cycle: This is an optional backroom cycle concerned with product integration, or manufacturing and distribution of any products required for the delivery cycle.
     – a delivery cycle: This is the actual delivery of the deliverable to the use. In other words, a delivery cycle contains the initial operational implementation of an Evo step, and its handover to stakeholders. It involves implementation activities such as training, installation and field-testing. The type and size of system change involved in a delivery cycle can vary, but is usually subject to project-defined step constraints on resource utilization. Usually, both financial cost and delivery frequency must be between 2% and 5% of project total budgets for cost and time respectively.
5. Result cycles, for different steps, can be executed serially and in parallel. The reason for this is the variable times taken for implementation (specifically development and production cycles) and the Evo requirement to achieve a reasonably short delivery cycle frequency. For example, the average delivery cycle frequency could be stipulated to be weekly or monthly, but a specific result might take six months from initiation to actual result delivery, due to such factors as research cycles, order cycles, construction cycles and approval processes. These processes would normally be sought to be done *in parallel* with other Evo cycle activities, so

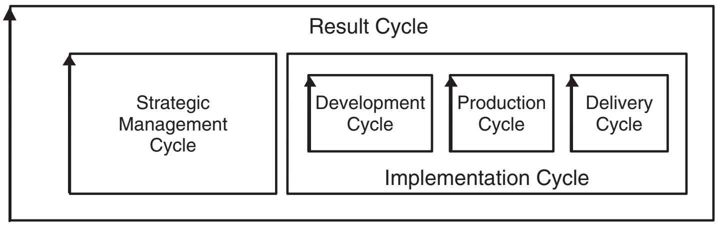


Figure G22
Diagram shows the component cycles of a Result Cycle.

that the Evo management team and their stakeholders would still experience some result delivery within the stipulated delivery cycle time.

6. The development and production cycles are termed 'backroom' activities and the delivery cycle is termed a 'frontroom' activity. One useful analogy is to think of the way in which a restaurant delivers to its customers. Ideally, delivery to the table is independent of food and drink preparation times!

Synonyms: Result Production Cycle; Step Cycle.

Related Concepts: Delivery Cycle *049; Development Cycle *413; Implementation Cycle *123; Production Cycle *407; Strategic Management Cycle *408.

Review **Concept *197**

A review is any process of human examination of ideas with a defined purpose and defined standards of inquiry.

Notes:

1. A 'go/no-go review' is a particular type of review for giving approval, or not, to a particular plan or idea.
2. Reviews should always have the benefit of the specification under review having successfully exited SQC processes using both Specification Rules and Specification Review Rules. Such SQC processes determine a specification's objective craftsmanship quality (conformance to standards). For example, a major review with

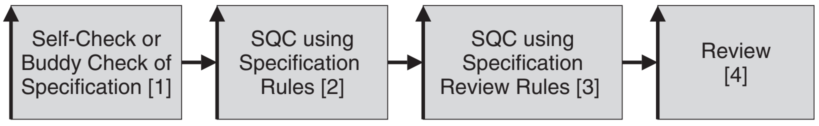

Self-Check or Buddy Check of Specification [1]
SQC using Specification Rules [2]
SQC using Specification Review Rules [3]
Review [4]

Figure G23
The diagram shows Review following SQC processes. The sequence of SQC processes leading to review(s) is as follows:

- Before any SQC is carried out, informal checking might be carried out by the specification writer or by a colleague [1]
- SQC (a more formal team process) is carried out by a group of people checking the specification against specification rules [2]
- If successfully SQC exited, further SQC can be carried out using specification review rules. This is to check the validity of entering a review process by carrying out a number of pre-checks using the relevant review criteria (types of review include Architecture Review and Business Review.) [3]
- If successfully SQC exited, a review can be carried out to decide on future actions [4].

financial consequences should not proceed if the estimated remaining major defects/page for a specification are, at entry, any greater than 1.0.

Related Concepts: Specification Quality Control (SQC) *051; Specification Rules *129; Specification Review Rules *543.

Risk **Concept *309**

A risk is any factor that could result in a *future negative* consequence. A Risk parameter can be used to specify known risks.

Notes:

1. Negative results are results that are worse than required, planned, or expected.
2. Examples of risk factors include:
   - lack of information about a design idea
   - inappropriate information about a design idea.

Source: See Bernstein's book on the history of risk (Bernstein 1996). One prominent economist (Knight) wanted to distinguish risk from uncertainty, in the sense that risk was measurable (Bernstein 1996, Page 219). Knight was also skeptical as to whether past data was sufficiently like a specific unique instance, and sufficiently detailed, to tell us what the probability of a future event would be.

Synonyms: Threat *309.

Related Concepts: Uncertainty *310; Safety Factor *131.

Role **Concept *253**

A role is a defined responsibility, interest or scope for people.

Related Concepts: Role [SQC] *411; Stakeholder *233.

Rule **Concept *333**

A rule is any statement of a standard on how to write or carry out some part of a systems engineering or business process.

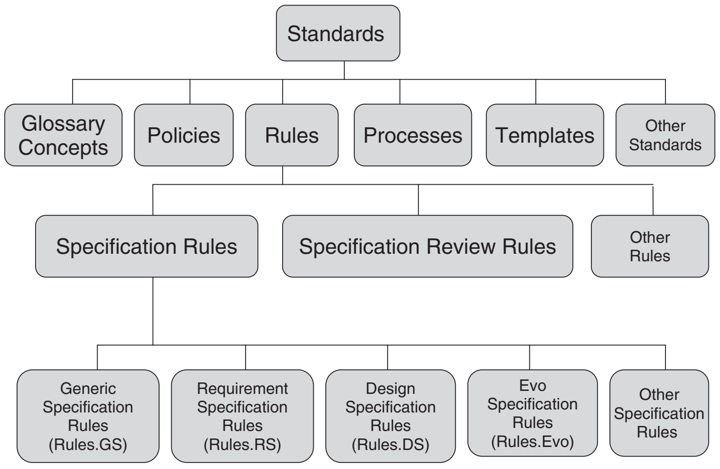

Standards
Glossary Concepts
Policies
Rules
Processes
Templates
Other Standards
Specification Rules
Specification Review Rules
Other Rules
Generic Specification Rules (Rules.GS)
Requirement Specification Rules (Rules.RS)
Design Specification Rules (Rules.DS)
Evo Specification Rules (Rules.Evo)
Other Specification Rules

Figure G24
Rules as standards. Some of the different types of Specification Rules are shown.

Notes:

1. Numerous different types of rule exist, including business rules, system rules and design rules.

Related Concepts: Standard *138; Specification Rule *129; Specification Review Rule *543.

Safety Deviation — Concept *405

A safety deviation is a measure of the estimated-or-observed difference between a required safety margin, and the estimated or actual system attribute level. 'How safe?' or 'How safe compared to plan?'

Notes:

1. Each scalar attribute target (performance goals and resource budgets) will potentially have its own computable value for safety deviation.
2. Safety deviation expresses how far away the current design proposal, or Evo step implementation, is estimated to be from the desired safety level. The higher the negative deviation is, the greater the 'risk of failure' to deliver the target level of the attribute.
3. The safety deviation can be use by technical management to monitor the progress of a design or a real evolving project delivery. Management will need some policy regarding setting and respecting safety factors. They will need to set some standards regarding the degree to which designs include planned safety factors. This is a specific tactic for risk management.
4. When using the Impact Estimation method and a spreadsheet model, the safety deviation computations can normally be done automatically using the values of the requirements and the design impacts. Automatic warning of insufficient safety is a possibility.

Related Concepts: Safety Factor *131; Safety Margin *637.

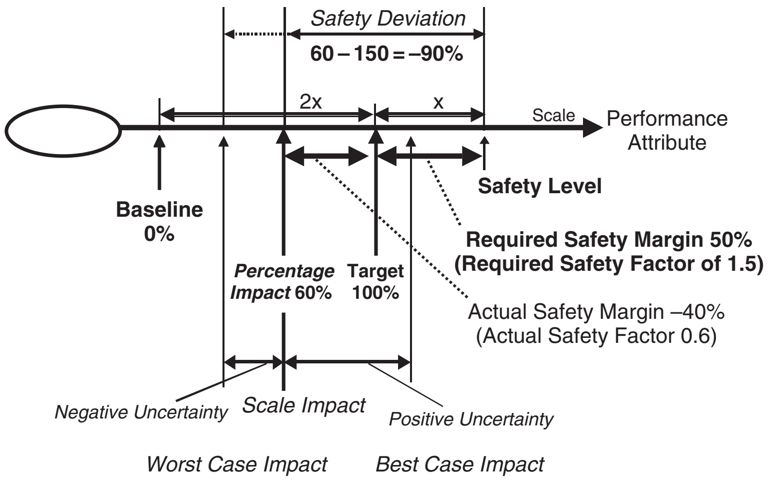


Figure G25

Diagram showing calculation of Safety Deviation for a quality objective. The safety margin is relative to the target level and the distance between the baseline and the target. In this example, as the required safety margin is 50%, it must be 50% beyond the target level. The distance (x) is worked out from the distance between the target and the baseline (2x).

Safety Factor **Concept *131**

A safety factor is the dimensionless ratio of 'conscious over-design' that is either required, or actually applied to some part of a system.

Notes:

1. A safety factor is used to communicate about risk. It is used to ensure that the design compensates adequately for both systems engineering and operational uncertainties.
2. Historically, safety factors were applied to mechanical loads. We are using it here to describe the amount of safety margin we wish to have designed into the system. The target and constraint levels are specified at the required levels and then the safety factor is applied to allow safety margins. (An assumption is being made here that there is only one safety factor involved; there could be several.)
3. A safety factor is either prescribed by standards, such as engineering rules or policy, or it is specified at project level.
4. A safety factor is a dimensionless ratio. Compare to a safety margin, which is either expressed using units of measure (as it is the difference between two levels on a Scale), or as a percentage value based on the required target or constraint level being 100%.

   Example:

Required Level	*Estimated/Actual Level*	*Estimated/Actual Safety Factor*	*Estimated/Actual Safety Margin*
100%	100%	1	0%
100%	50%	0.5	−50%
100%	200%	2	100%

   This example assumes no specific safety factor has been set. It calculates the estimated/actual safety factor and safety margin based on the required level being 100%.
5. If we want to explicitly specify a safety factor, we can do so in a variety of ways using the Safety Factor parameter.

Synonyms: Safety *131.

Related Concepts: Safety Deviation *405; Safety Margin *637.

Keyed Icon: nX "Where n is the numeric safety factor."

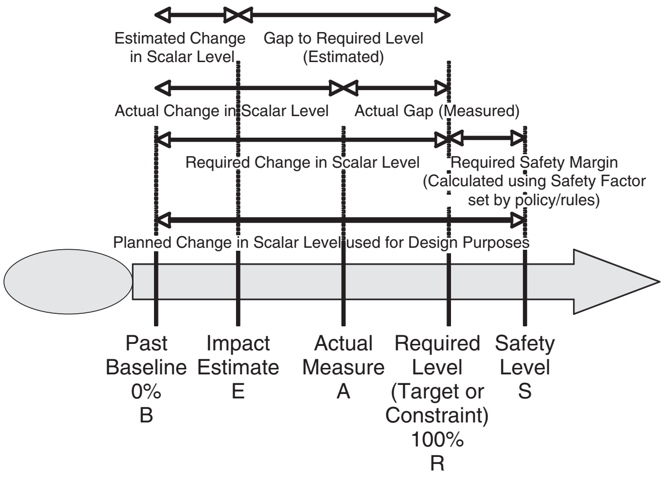
Example:
Safety Factor: 3X.

Safety Margin **Concept *637**

A required safety margin is a scalar difference between a required defined target or constraint level, and its calculated safety level derived using the appropriate safety factor.

Notes:

1. An estimated or actual safety margin can also be calculated. If no specific required safety factor is specified then the safety margin can be calculated relative to the (estimated or actual) target or constraint level (100%, equivalent to a safety factor of 1).
2. A safety margin can also be expressed as a percentage based on the target or constraint level being 100%, and the baseline being 0%.


Estimated Change in Scalar Level
Gap to Required Level (Estimated)
Actual Change in Scalar Level
Actual Gap (Measured)
Required Change in Scalar Level
Required Safety Margin (Calculated using Safety Factor set by policy/rules)
Planned Change in Scalar Level used for Design Purposes
Past Baseline 0% B
Impact Estimate E
Actual Measure A
Required Level (Target or Constraint) 100% R
Safety Level S

3. A ‘safety factor’, by comparison, is a dimensionless ratio.

Related Concepts: Safety Factor *131; Safety Deviation *405.

Scalar **Concept *198**

Scalar is an adjective used to describe objects, which possess or are measured using at least one scale of measure.

Notes:

1. Performance and resource attributes are scalar.
2. Scalar attributes, provided they are not elementary (one Scale only), can have numerous scales of measure (A complex scalar attribute will possess more than one elementary attribute.)
3. All numeric levels on scales of measure can be described as *scalar* values.
4. A scalar object can be contrasted to a binary object, which is not scalar, but is in one of two states (commonly, either present or absent).

Related Concepts: Scale *132; Binary *249.

Scale **Concept *132**

A ‘Scale’ parameter is used to define a scale of measure. All elementary scalar attribute definitions require a defined Scale.

A Scale states the fundamental and precise operational definition for a specific scalar attribute. It is used as the basis for expressing many of the parameters within the scalar attribute definition (for example, Meter, Goal and Budget): all scalar estimates or measurements are made with reference to the Scale. The Scale states the units of measurement, and any required scalar qualifiers.

Notes:

1. A Scale is not a numeric level along the defined Scale (it is not a benchmark, target or constraint).
2. A Scale is not the measuring instrument (*that* is specified by the Meter parameter).

   A Scale describes something, which is variable, trackable, observable and countable in nature. A Meter specification is the definition of the *means* of measuring the 'level of capability' expressed by a Scale. For example, a Meter is the 'voltmeter' for measuring on a Scale of 'volts.' A Scale is abstract, while a Meter is a real-world, practical means of obtaining measurements. A specific Meter itself has multiple performance and cost attributes. These are the basis for selecting a particular Meter.

There is a big distinction between units of measure (Scale) and a measuring tool (Meter or Test). Units of measure can be used to express clear ideas, like requirements, quite independently of the possibilities and problems of measurement itself. I can express a clear idea, "I want to get to the moon and back in one second," quite clearly. The fact that I cannot really do it, or measure it, is beside the point. I stress this because I have discovered that many people waste their energy arguing against a particular *quantification*, when all their arguments are only related to the difficulty of its accurate *measurement.*

3. Many Scales are specified as 'generic scales.' A generic scale is a Scale that requires final specification of 'scale qualifiers' (in the Scale definition) by means of 'scale variables' (in a target or constraint specification), in order to have an operationally precise definition.

   Example:
   Scale: Time to Master defined [Tasks] by defined [Learner Type].
   There are two scale qualifiers in the above generic scale definition, which require definition by scale variables. For example, 'Goal [Tasks= Update, Learner Type= Novice]: 30%'.

"To leave [soft considerations] out of the analysis simply because they are not readily quantifiable or to avoid introducing 'personal judgments,' clearly biases decisions against investments that are likely to have a significant impact on considerations as the quality of one's product, delivery speed and reliability, and the rapidity with which new products can be introduced."
<- *R. H. Hayes et al.* Dynamic Manufacturing*, Free Press 1988 NY Page 77,* quoted in Mintzberg (1994 Page 124)

"Aligning Rewards with Measurements 'You have to get this one right. . . . a universal problem: What you measure is what you get – what you reward is what you get. Static measurements get stale. Market conditions change, new businesses develop, new competitors show up. I always pounded home the question 'Are we measuring and rewarding the specific behavior we want?'"
<- *Jack Welch, former CEO* General Electric *(Welch 2001 Page 387)*

Synonyms: Scale of Measure *132.
Related Concepts: Scale Qualifier *381; Scale Variable *446; Meter *094.
Keyed Icon: -|-|-

Scale Impact Concept *403

For a scalar requirement, a scale impact is an absolute numeric value on the scale of measure. It can be an estimated value, or actually achieved, measured value. It is the level estimated or achieved if a specified design idea (or set of design ideas or Evo step) is implemented.

Notes:

1. In an Impact Estimation table, Scale Impact is customarily used together with Percentage Impact, as alternative views of the impact estimate. We use Scale Impact when we just want to know the real final result, which

includes the effects of implementing all previous designs. We use Percentage Impact when we want to understand the effect in relation to moving from the baseline towards the goal. In other words, Scale Impact is an absolute numeric value, while Percentage Impact is a relative value dependent on the Scale Impact, and the Baseline to Target Pair.

2. Care has to be taken, as the impact of a design idea varies, depending on the system technology it is added to and used in. In other words, the impact of a design idea is not a constant, irrespective of the circumstances it is implemented in. There can be dependencies and interactions. Altering the order of implementing design ideas could affect the immediate level of impact of any specific design idea. However, given that the choice is usually just 'what shall we implement next on a specific system?' it is not necessary to assess the impacts of all the valid design idea combinations.

Synonyms: Absolute Impact *403.

Related Concepts: Incremental Scale Impact *307; Percentage Impact *306.

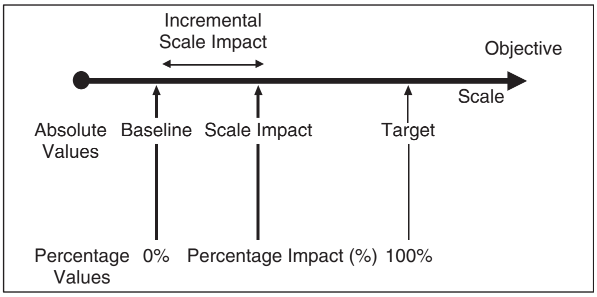


Figure G26

Scale Qualifier **Concept *381**

A scale qualifier is a term within the *definition* of a Scale parameter. It specifies the need for a qualifier condition with an assigned scale variable to be specified when referencing or applying the Scale in another statement. For a given Scale, any useful number of scale qualifiers can be defined.

Example:
Scale: Time to learn a defined [Task]. "Task is a scale qualifier."

Scale qualifiers are generic; each scale qualifier needs to be explicitly assigned a corresponding 'scale variable' (unless a default is being used) when the Scale is used in other parameter statements (such as any benchmarks or targets).

Example:
Goal [Task = Setup]: 10 minutes. "Setup is a scale variable defining the scale qualifier, Task that was defined in the previous example."

The purpose of scale qualifiers is to allow a scale specification to be more generalized and flexible; this consequently makes a scale specification more *reusable*.

Notes:

1. A scale qualifier is expressed and specified by enclosing the qualifier condition in square qualifier brackets. The word 'defined' is optionally

specified immediately prior to the square brackets, to help emphasize that a more specific definition needs to be provided when the Scale is referenced, for example by a Goal statement.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].

2. A *default option* can be specified in order to make explicit specification unnecessary.

Example:
Scale: The defined [Time Units: *Default = Hours*] needed to do a defined [Task] by a defined [Employee Type].

3. The scale qualifier parameter can also be 'referenced' by using the *same sequence* as used in the scale definition: Note in this example, an additional qualifier condition, not in the original scale definition, has been added. This is OK. You can add any number of additional conditions that you want.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].
Goal [Hours, Answering Help Desk Queries, Experienced, Country = Finland]: 60 Hours.

Or, an *explicit* reference to the scale qualifier tag ('Time Units = Hours') may be made, for increased clarity.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].
Past [Time Units = Months, Task = Complaint Handling, Employee Type = Supervisor]: 6 Months.

4. The sequencing of scale qualifiers and scale variables is not critical as long as the parameters are unambiguous to the specification user.

Synonyms: Scale Parameter *381; Embedded Scale Qualifier *381.
Related Concepts: Qualifier *124; Scale Variable *446.

Scale Uncertainty — Concept *143

A scale uncertainty is an estimate of the error margins for a specific scale impact (that is, it provides information about the plus-and-minus range on the scale of measure over which an estimate for a scale impact can possibly vary). It allows calculation of the best and worst case borders.

Notes:

1. Experience data should be used for guidance, and specified as evidence together with its source(s).
2. In some cases, the error margins may not be symmetrical about the main estimate. It may be appropriate to use only the more extreme uncertainty value, or to specify the asymmetry directly using the two different numbers (for example, +30% and −40%).

Related Concepts: Scale Impact *403.

Scale Variable — Concept *446

A scale variable is the specific term assigned to 'finally' define a qualifier condition for a scale qualifier.

If we fail to explicitly define a scale variable in a particular statement and there is a defined *default* scale variable, then the defined default (… for defined [Tasks: Default = Update] …) will be assumed to be the intended scale variable for that statement. If there is no default defined, then the statement, in this instance, is defective.

Example:
Responsiveness:
Scale: Number of Days after a defined [Day] until enquiries answered.
Goal [Day = Monday]: 3.
'Day' is the scale qualifier. Monday is a constant, assigned as the relevant scale variable, out of the set of the possible days of the week.

Example:
Excess Speed:
Scale: Kilometers per hour in excess of the defined [Maximum Speed: Default = Posted Legal Limit Speed].
Fail [Maximum Speed = 80]: 0 Kilometers per hour.
You would have to finally determine the definition in real driving conditions.

Example:
Scale: The time needed for a defined [Task] by defined [People] in defined [Places].
Goal [Update, Naval Officer, At Sea]: 20 minutes.
Or, alternatively, using explicit references to the scale qualifiers, (Task = …).
Goal [Task = Update, People = Naval Officer, Places = At Sea]: 20 minutes.
'Update', 'Naval Officer', and 'At Sea' are scale variables, defining one of the three scale qualifiers (Task, People and Places). The scale variables are also specification variables because we don't really know what they mean until we look at their definition. For example, what if 'At Sea: Defined As: In any craft which floats on any type of water'? Does that include wooden model boats in a small pond?
Related Concepts: Scale Qualifier *381; Specification Variable *456.

Scope — Concept *419

A 'Scope' describes the extent of influence of something. Scope can apply to anything, like a specification, or a specified system or project. The 'extent of influence' can be described in any useful terms. This includes using any Planguage expressions or parameters. For example, any [time, place, event] qualifier conditions, and any other parameters, such as 'Stakeholders', can define the extent of influence of a specific specification within the system scope.

Notes:

1. There are two especially useful notions of scope:
   - Global Scope: global scope specifications (potentially) influence or dictate something (like a constraint) to all areas of a defined system, unless some overriding or higher-priority specification cancels its influence. For example, 'Project Scope' (Hooks and Farry 2000 Pages 43–58).
   - Local Scope: a local specification is unable and unwilling to influence or determine specifications (such as requirements and designs) beyond a defined sub-system area.

Related Concepts: View *484; Context *483; Qualifier *124; Time *153; Place *107; Event *062.

Example:

Scope [Project X]: USA Parent Market only.

An example of an explicit Scope specification: Scope can otherwise be indicated by a qualifier and by many other parameters (such as Authority for example).

Side Effect — Concept *273

An impact by a design idea, on any requirement attribute, other than the direct impact(s) we primarily intended.

Notes:

1. Side effects can be evaluated at a design stage and/or observed at an implementation stage, or even operational or decommissioning stage. Conventional usage of 'side effect' implies 'negative effects,' but positive side effects can be just as likely, and just as interesting!
2. Side effects can be of the following categories:
   - 'Intended or unintended': 'Intended' means that we have chosen the design because we knew about and valued those particular side effects;
   - 'Known or unknown': 'Known' means we were aware of the existence and possibly the levels of the side effects. 'Unknown' means we were not initially aware of the side effects, but may have become aware of them at some later stage of considering the design (such as in testing, in a review or in operation);
   - 'Negative, neutral or positive'.

Related Concepts: Impacts *334: This parameter is used to specify side effects.

Software Engineering — Concept *572

Software engineering is the discipline of making software systems deliver the required value to all stakeholders.

Notes:

1. Software engineering includes determining stakeholder requirements, designing new systems, adapting older systems, subcontracting for components (including services), interfacing with systems architecture, testing, measurement and other disciplines. It needs to control computer programming and other software related sub-processes (like quality assurance, requirements elicitation, requirement specification), but it is not necessary that these sub-disciplines be carried out by the software engineering process itself. The emphasis should be on control of the outcome – the value delivered to stakeholders, not of the performance of a craft.
2. The concept 'required value' (above) is used to emphasize the obligation of the software engineer to determine the value or results truly needed by the stakeholders, and not to be fooled by omissions, corruptions and misunderstandings of the real-world value.
3. The concept 'all stakeholders' (above) is used to emphasize the broad range of internal stakeholders (like the development project and the producing organization), and external stakeholders (such as users,

customers, governments, add-on suppliers) that the software engineering process must be obliged to deal with. We are consciously trying to break away from older, narrower notions that software engineering is all about satisfying users or customers alone.

Related Concepts: Software *570; Engineering *224; Systems Engineering *223; Stakeholder *233; Value *269.

Source — Concept *135

'Source' is a synonym for process input *information* (as opposed to process input *materials*).

Notes:

1. Source specifications used in SQC, are contained in documents that are usually of earlier production, and probably at higher levels of authority, global scope and abstraction. For example: contracts are sources for requirements. Requirements are a source for design. Requirements and design are sources for Impact Estimation. Design is source for planning and construction or programming. Older specifications and change requests are sources for updated specifications.

Related Concepts: Evidence *063.

Keyed Icon:

Specification — Concept *137

A 'specification' communicates one or more system ideas and/or descriptions to an intended audience. A specification is usually a formal, written means for communicating information.

Notes:

1. A specification is usually written, but it could be oral.
2. The term 'specification' can refer to a single element of a larger specification or to a larger set of specifications. It includes the entire set of parameters and lines of text needed to specify an idea.
3. The specification concept can deal with past, present and future; it is not confined to requirement or design specification.
4. There are many classes of 'specification' including {requirements, design analysis (such as Impact Estimation tables), and project plans (such as Evo plans)}.
5. 'Specification' can be described as a class of document that is used to control the outcome of a project.
6. The term 'specification' is often specifically intended to refer to project specifications, sometimes popularly called 'specs.'
7. Specification can be categorized as Commentary or Non-Commentary. Non-Commentary consists of Core Specification and Background Specification. This categorization recognizes the significance of the specification content. For SQC purposes, it is important to make this distinction, as finding major defects in the Core Specification is the key task.

Abbreviations: Spec.

Related Concepts: Definition *044; Description *416; Documentation *579; Document *180; Planguage Concept *188; Commentary *632; Non-Commentary *294; Background *507; Core Specification *633.

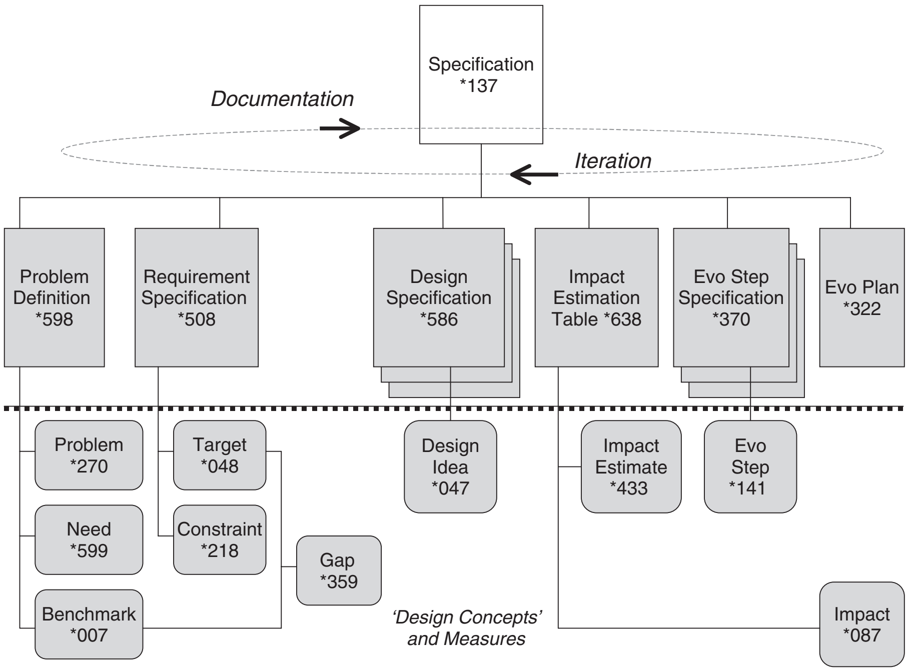


Figure G27
Different kinds of specification.

Specification Quality Control **Concept *051**

Specification Quality Control (SQC) is a rigorous specification quality control discipline. SQC is concerned with defect detection, defect measurement, defect removal, process improvement and entry/exit controls. It is based on evaluating specification conformance to specification rules.

Notes:

1. During SQC, specifications are checked against their relevant rules, sources and kin documents for validity. Any rule violations are defects. The density of defects is used to judge the 'quality' of craftsmanship of the specification.
2. SQC includes the Defect Detection Process (DDP) and the Defect Prevention Process (DPP). Both are defined in detail in (Gilb and Graham 1993). When the DPP (process improvement) is used the scope goes beyond quality control and extends to quality assurance.
3. Traditionally, SQC does not pretend to judge the specifications in terms of their relevance or profitability in the real world. It is primarily concerned with making sure that the specifications are clear, complete and consistent by checking a specification and any of its source and kin documents against Specification Rules. It judges whether the specification is suitable to be used in subsequent engineering or management processes. However, by using a different type of rules, Specification Review Rules, it is possible to extend the SQC process to checking the readiness of specifications for review. This could be for a business review or a technical review. See Review *197.

Description: Chapter 8, "Specification Quality Control: How to Know How Well You Specified."
Acronym: SQC *051.
Synonyms: Inspection *051; Peer Review *051; "For additional specialized synonyms, see (Wheeler, Brykczynski and Meeson 1996)."
Related Concepts: Quality Control *279; Specification Defect *043; Specification Rule *129; Specification Review Rule *543; Review *197.

SQC **Concept *051**

Acronym for 'Specification Quality Control'.

Stakeholder **Concept *233**

A stakeholder is any person, group or object, which has some direct or indirect interest in a system. Stakeholders can exercise control over both the immediate system operational characteristics, as well as over long-term system lifecycle considerations (such as portability, lifecycle costs, environmental considerations and decommissioning of the system).
The parameter 'Stakeholder' can be used to specify one or more stakeholders explicitly. We can attach stakeholder information to any elementary specification, or to a set of specifications, as appropriate.

"4.16 Stakeholder: An interested party having a right, share or claim in the system or in its possession of qualities that meet their needs."
Draft Standard ISO/IEC 15288 (ISO/IEC 1999)

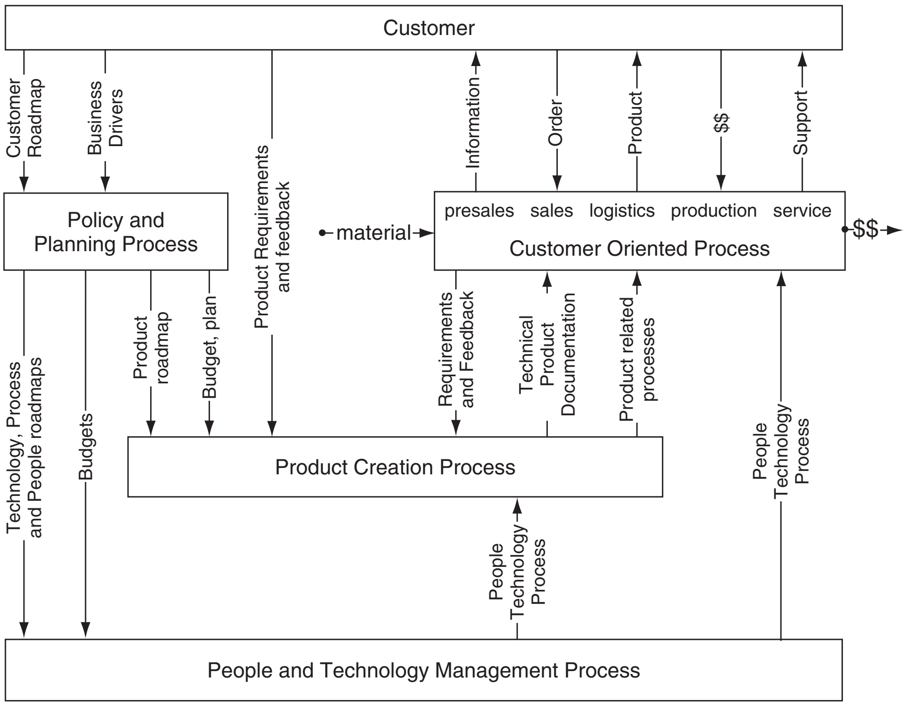

Customer
Customer Roadmap
Business Drivers
Product Requirements and feedback
Information
Order
Product
$$
Support
Policy and Planning Process
material
presales sales logistics production service
Customer Oriented Process
$$
Technology, Process and People roadmaps
Budgets
Product roadmap
Budget, plan
Requirements and Feedback
Technical Product Documentation
Product related processes
People Technology Process
Product Creation Process
People Technology Process
People and Technology Management Process

Figure G28
Some stakeholder concepts. Courtesy Gerrit Muller, Philips, Eindhoven, NL. (Muller 1999).

Notes:

1. The views and needs of stakeholders have to be sought and listened to. For example, stakeholders might have an interest in:
   - setting the requirements for a process, project or product
   - evaluating the quality of a product
   - using the product or system, even indirectly
   - avoiding problems themselves as a result of our product or system
   - the system or product being compatible with another machine or software component
   - determining the constraints on development, operation or retirement of the system or product
2. Stakeholders specify requirements, directly or indirectly, for the system attributes (function, performance, resource, design constraints, and condition constraints). They determine the degree of product or system success or failure.
3. Systems engineers should determine which requirements the stakeholders need, and which requirements they can afford. Even if the stakeholders are not currently conscious of those needs and limitations!

   Example:

   Goal [Stakeholders = {Installers, Service People}, End This Year]: 60 hours <- Marketing Authority.

   Marketing Authority: Stakeholder: Our Service Organization.

   The Goal requirement applies to a set of defined stakeholders. The requirement authority (the one who has requested this Goal level) is defined as another stakeholder.
4. Stakeholders can be internal or external to a system – it depends on the context. Internal stakeholders are typically in our development organization. External stakeholders might be the users and customers of the developed system. Often very external stakeholders are instances like laws and government organizations that can impose requirements on our system. This distinction is useful:
   - to help us develop better lists of stakeholders
   - so we don't get fixated on the 'customer/user' as the only requirements source
   - to give us a systematic set of (internal) stakeholders to deliver to, as we evolve the system, even when it is not ready for external stakeholders.

Related Concepts: Owner *102; Client *235; Sponsor *396; Decision-Maker *237; Consumer *038; User *234; Designer *190.

Standards — Concept *138

A standard is an official, written specification that guides a defined group of people in doing a process. It is a best-known practice.

> "A thing serving as a recognized example or principle to which others conform or should conform or by which the accuracy or quality of others is judged."
>
> *Oxford Dictionary*[3]

[3] *The New Shorter Oxford English Dictionary*, 1993. Oxford: Oxford University Press. ISBN 0-19-861134-X.

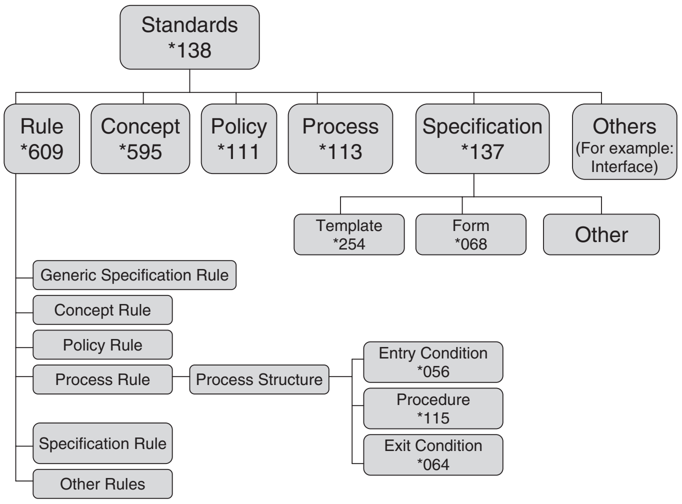


Figure G29
Shows a variety of work process standards provided by Planguage to help define work processes.

Standards include: {rule, policy, process, entry condition, procedure, exit condition, form, template}.
Synonyms: Work Process Standards *138.
Related Concepts: Rule *129; Policy *111; Process *113; Form *068; Template *254.

Status **Concept *174**

Status is the outcome of an evaluation of a defined condition (or set of conditions). Status can be a matter of establishing true/false or it can be a set of different indicators (status settings). Status determines whether a specification or system component applies/is usable or not.
Notes:

1. For a specification, an evaluation of its status is done whenever a qualifier or a conditional statement is evaluated. Status is implied.
   Example:
   Goal [By End of Year, USA, Manager, Product X, If Customer Y Signed]: 500 Items.
   The Goal only applies (500 Items is a Goal) if all the qualifying [time, place and event] conditions are met (Status = true).
2. An explicit Status specification can also be made.
   Example:
   Safe: Status: {A, B, C, Not D, Weekday}.
   A: Condition: Everybody feels good and no one panics.
   B: Event Condition: No official alarm is raised by Building Safety, or on public address.
   C: Indicator Condition: No red light flashing on your workstation to warn of unsafe air conditions.
   D: Sign Condition: Lobby Sign says “No Smog.”

Different types of Condition are examined. Status is used to collect the results of the condition set. The reason you might organize things this way is to provide clarity of specification and to enable reuse of specifications. For example 'No Alarms: Status: {B, C}.'

3. Status is commonly explicitly used as a parameter for classifying *specification status.*
   (The underlying conditions are usually not explicitly specified with a specification status.)
   Example:
   Document XYZ: Version: February 22, 2005. Status: Draft.
   This can be used for a document, or any defined specification, including a single requirement or design specification. Suggested status settings (indicators) include:
   - Undetermined
   - Under Revision
   - Exited
   - Approved
   - Validated
   - Verified (proven to be present and correct by some form or test or observation)
   - 'Initial, Defined, Agreed Upon, Released' for 'working state' (as used by Daimler Chrysler, 2002 [Personal Communication 2002]).

   Rationale:
   - to clearly warn specification readers when a specification is not really approved for certain uses.
   - to allow even small subsets of a larger specification document to be independently upgraded or downgraded in status.
   - to help control the evolution of any technical specification.
4. Status can also be used for *system component control.* For example, at the beginning and end of a process, the relevant entry and exit conditions respectively are usually status-checked. Each entry/exit condition could have a true or false status.

Synonyms: State *174.
Related Concepts: Condition *024.

Stretch Concept *404

A Stretch parameter is used to define a *somewhat more ambitious* target level than the *committed* Goal or Budget levels.

A Stretch level is specified on a defined Scale, under specified conditions [time, place, event]. There is *no commitment* to deliver a Stretch level. Stretch announces that there is *some* stakeholder value at that level, *if* we can find a *practical* or *economic* solution for delivering it.

Notes:

1. The intention is that a Stretch target is challenging, even quite difficult to attain.
   It is used in an attempt to *inspire and motivate* people to do their very best and to do something more than they would *otherwise* dare to do.
2. There is *not a project commitment* to attain a Stretch target. The technology to reach it may be unknown or unavailable. The technology could be too expensive at present to make it profitable to make

this level the Goal or Budget level. However, if without using any extra resources, the project could reach a Stretch level, it would be welcomed. It would have some potential stakeholder value as a result.
3. A Stretch level specification can be a resource target or a performance target.

"Stretch is reaching for more than what you thought possible.... In a stretch environment, the same field team is asked to come in with 'operating plans' that reflect their dreams – the highest number they think they had a shot at: their 'stretch'. The discussion revolves around new directions and growth, energizing stuff.... We'll never stop 'stretching'."
Jack Welch former CEO General Electric, in Jack: Straight from the Gut
(Welch 2001 Pages 385–6)

Synonyms: Stretch Target *404; Stretch Level *404 (see Level *337).
Related Concepts: Need *599; Target *048; Goal *109; Budget *480; Wish *244.
Keyed Icon: >+ In context: --->+--->O--->+--->
Historical Note: The Stretch concept was first used in Planguage by Pete Fuenfhausen, then at Nokia, Dallas, TX, September 1999.

Supports — Concept *415

'Supports' is used to indicate what an attribute is *mainly intended to* support. It differs from 'Impacts,' which can include information about all the negative unintended side effects. 'Supports' only lists selected intended main supporting impacts.
Example:
Low RF Power Output [Radio Heads]:
Supports: {Availability, Co-existing, Robustness, Others} <- Marketing Specification 4.2.1.8.
Related Concepts: Impacts *334; Is Supported By *414.

Survival — Concept *440

Survival is a state where the system can exist. Outside the survival range is a 'dead' system caused by a specific attribute level being outside the survival range. For example, 'frozen to death' or 'suffocated.'
A Survival *parameter* specifies the upper or lower acceptable limits under specified conditions [time, place, event], for a scalar attribute. It is a *constraint* notion used to express the attribute levels, which define the survival of the entire system.
For example, a system violating a Survival limit becomes illegal, or totally unprofitable, or in strong violation of a contract. Survival limits are typically derived from laws, regulations and contractual specifications.
Each survival specification should always have a clearly stated Authority or Source specified.
Notes:
1. Survival is used to clearly state the nearby existence of a strong 'sudden death' borderline for an entire system. Worse than a Survival level is a 'catastrophe.'

2. One elementary scalar requirement can have several simultaneous Survival specifications. This is because there can be different qualifiers for each Survival level (that is, different times, places and events can apply). For example, different stakeholders can set different criteria.
3. Survival can be used to set resource limits or performance limits, at both extremes (---[--- and ---]--->O---[--- and ---]--->).
4. Survival implies strong authority behind it (like a Law or Corporate Policy). You should always document the exact source of this authority, using Source and/or Authority. You should also include any other information, which that would help the specification reader to understand why this requirement has been classified as a Survival Limit (such as Rationale).
5. A Survival Limit violation will *not necessarily* lead to *real* catastrophic failure. The failure degree depends on discovery and reaction from the Authority behind it at the time and place of violation. For example, just because the heart stops does not mean the person is finally dead. However, the heart cannot be expected to start up on its own: death may well result.

Example:
Financial Cost Budget:
Scale: Cost in $ for Total Project.
Budget [Lifetime Warranty]: $9.5 million. Rationale: To allow for risks and any lawsuits.
Fail [By Contract Completion]: $9 million. Rationale: To ensure Profit Level.
Survival [By Contract Completion]: $10 million <- Contract 5.4.3.
Budget, Fail and Survival specify requirements with varying priority. Budget implies 'get to this level for success.' Fail specifies 'must reach this to avoid any failure (disappointment in the results).' Survival sets the upper financial limit to avoid disaster.

Synonyms: Survival Level (see Level *337); Survival Limit (see Limit *606).
Related Concepts: Fail *098; Limit *606.
Keyed Icon: [and/or] "The '[' being a lower limit and the ']' being an upper limit."

Systecture © — Concept *564

See Systems Architecture *564. Systecture is a conjunction of the terms 'systems' and 'architecture'.
Historical Note: In July 2002, in connection with a book manuscript on systems architecture, I needed a catchy term for the book title. In my 1988 book, Principles of Software Engineering Management, *I had coined the terms 'softecture' and 'softect'. So, it seemed natural to extend this to the system engineering area. A web search turned up* www.systect.com *(Systect, Inc. 'The system architects') a systems architecture company, but no use of Systecture at all.* © Tom@Gilb.com *2002. Permission is granted to use the term as a generic word. I felt there was a need to get away from the 'architecture' term. Architect is from 'Archi-Tecton,' which means 'Master Builder.' 'Archi' is not from 'Arch', but from 'Arche': primitive, original, primary.*[4]

[4] Contributed by Niels Malotaux.

System [Planguage] ***Concept *145***

A system is any useful subset of the universe that we choose to specify. It can be conceptual or real. In Planguage, a system can be described fundamentally by a set of attributes. The attributes are of the following types:

- function: 'what' the system does
- performance: 'how good' (quality, resource saving, workload capacity)
- resource: 'at what cost' (resource expenditure)
- design: 'by what means.'

In addition, other factors describing various aspects of the system can be specified. These include:

- requirements
- dependencies
- risks
- priorities.

All these specifications (the attributes and the additional factors) are qualified by time, place and event conditions.

Notes:

1. There are specific Planguage parameters for capturing all the system information, including: Function, Performance, Resource, Design, Requirement, Dependency, Risk and Priority.
2. A Norwegian professor client of mine said (about 1969) that he detested the word 'system' because it "had the precision of the word 'thing'." I have ever since then been careful using it, and hope the Planguage definition limits the scope somewhat.
3. Here are some standard definitions of 'system':

   Standard Definition [System, ISO 9000, 2000]:

   "An object consisting of interrelated or interacting elements."

   Note, this ISO 9000 definition emphasizes the internal relationship or interaction of system elements. This has limited interest. The most central aspect of systems is how they are externally experienced and perceived by other systems, so the Planguage definition emphasizes the attributes and admits the possibility of all manner of description, including the 'interacting elements' – but chooses to emphasize real-world 'interaction' (between any one system and all others).

   Standard Definition [System, EIA/IS-731.1, 1996 Interim Standard]:

   "system: The aggregation of end products and enabling products that achieves a given purpose."

   Note in this EIA/IS-731.1 system definition, the concept of 'purpose' comes in. However, lost is the possibility of multiple stakeholders and multiple purposes through time.

   Standard Definition [System, ISO/IEC 15288, preliminary version 2000]:

   "4.17 System An object consisting of interrelated or interacting elements (ISO 9000: 2000).

   NOTE: In practice, a system is 'in the eye of the beholder' and the interpretation of its meaning is frequently clarified by the use of an associative noun, e.g. product system, aircraft system. Alternatively the word system may be substituted simply by a context dependent synonym, e.g. product, aircraft, though this may then obscure a system principles perspective."

Note in this ISO/IEC 15288 definition, the authors seem to see a problem with the concept, but try to solve it by encouraging specific adjectives to describe it. They stick to the official version [ISO] but do not mention attributes or purposes. A hint about systems principles is given. Standard Definition [MIL-STD 499B]:

"System: An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective."

Note this MIL-STD 499B definition is unnecessarily narrow (it does not include the Planetary system, or the molecular system, ☺) and unnecessarily broad (a people or product or process would be sufficiently narrow for many systems engineering purposes). It is good that it mentions the capability to satisfy requirements, but some systems have capabilities that satisfy nobody's requirements (like faults and side effects). Systems are as they are, whether we like it or not. We have to be able to understand and describe their attributes realistically, like them or not.

4. When defining a system, it is important to decide the relationship between the system being observed and changed, and the system of the people (the project) bringing about any change. Numerous different relationships can exist. At one extreme, a project can be completely within the system being modified. At another extreme, a project might be developing a product system to be sold into various, as yet unknown, target systems.

Synonyms: System *145; Object *099: A separate concept number has been allocated as the two terms tend to be used distinctly.
Related Concepts: Attribute *003.

Systems Architecture — Concept *564

Systems Architecture is the set of artifacts produced by Architecture Engineering. A systems architecture is a strategic framework and consists of models, standards and design constraints specifying mandatory and recommended best practice for implementing and maintaining systems.

Notes:

1. A systems architecture usually applies across a division or an entire organization.
2. A systems architecture varies in its level of detail depending on its maturity and what is required of it. Different organizational cultures will require different things. The main point is that a systems architecture should be cost-effective.
3. The aims of a systems architecture could include:
   - imparting technical strategy
   - sharing best practices
   - ensuring specific standards are adhered to (for example, security)
   - avoiding duplication of effort
   - reducing risk by promoting tried and trusted information
   - encouraging recognition and use of standard interfaces
   - promoting reuse

- ensuring compatibility of data structures amongst systems
- achieving economies of scale through standard platforms (especially for training, support and maintenance).

4. Individual systems will have their own architecture (Architecture *192), which will adhere to any relevant mandatory systems architecture.

Synonyms: Systecture *564.

Related Concepts: Architecture Engineering *499; Architecture Specification *617; Standards*138; Architecture *192.

Systems Engineering — Concept *223

Systems Engineering (SE) is an engineering process encompassing and managing all relevant system stakeholders requirements, as well as all design solutions, and necessary technology, economic and political areas.

The fundamental purposes of systems engineering are to:

- optimize the system solution at the highest level of stakeholder concerns,
- synchronize all contributing disciplines to contribute efficiently to the final system characteristics,
- consider the entire system life cycle needs,
- manage risks for the entire system and the entire system life.

An INCOSE Definition:

> "Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs."
>
> *(http://www.incose.org/whatis.html)*

Blanchard's Department of Defence (DoD) Definition:

> Systems Engineering is the "process that shall:
> 1. Transform operational needs and requirements into an integrated system design solution through concurrent consideration of all life-cycle needs (i.e., development, manufacturing, test and evaluation, verification, deployment, operations, support, training and disposal);
> 2. Ensure the compatibility, interoperability, and integration of all functional and physical interfaces and ensure that system definition and design reflect the requirements for all system elements (i.e., hardware, software, facilities, people, data); and
> 3. Characterize and manage technical risks."
>
> *(Blanchard 1997)*

An FAA (the USA Federal Aviation Authority) Definition:

> *Systems Engineering is:* "A hybrid methodology that combines policy, analysis, design, and management. It ensures that a complex man-made system or product, selected from the range of options available, is the one most likely to satisfy the customer's objectives in the context of long-term future operation or market environments.
>
> Systems engineering is applied throughout the system or product life cycle as a comprehensive, possibly iterative, interleaved, or recursive, technical process to*:*
>
> *a.* Translate an operational need into a configured system or product meeting the operational need
>
> *b.* Integrate the technical contributions of all available development resources, including all technical disciplines into a coordinated effort that meets established program cost, schedule and performance objectives. This involves a 'holistic view' (the design of the whole as

distinguished from the design of the parts). Such a view is multi-disciplinary in nature, rather than disciplinary or interdisciplinary;

c. Ensure the compatibility of all function and physical interfaces (internal and external)

d. Ensure that system or product definition and design reflect the requirements in system or product elements (outcome, hardware, software, facilities, people, and data).

e. Characterize [identify, define, and classify] technical risks, develop risk abatement approaches, and reduce technical risks by prevention and mitigation of impacts when risks are realized."

Source: FAA-iCMM Appraisal Method Version 1.0 A-19, INCOSE Conference CD, June 1999, Brighton UK (FAA 1998).

Notes:

1. The Systems Engineering process is a conscious attempt to avoid sub-optimal engineering. Without Systems Engineering, the success of the resulting system is more accidental than predictable. Systems engineering is necessary because there are so many possible places for product development to go wrong. For example, sub-optimal results might be caused by setting requirements for too narrow a list of stakeholders, or by using too narrow a set of design ideas to solve the problem of satisfying all project requirements. Another frequent problem, especially in well-established large companies, is for groups to produce optimal components yet produce a very sub-optimal complete system.

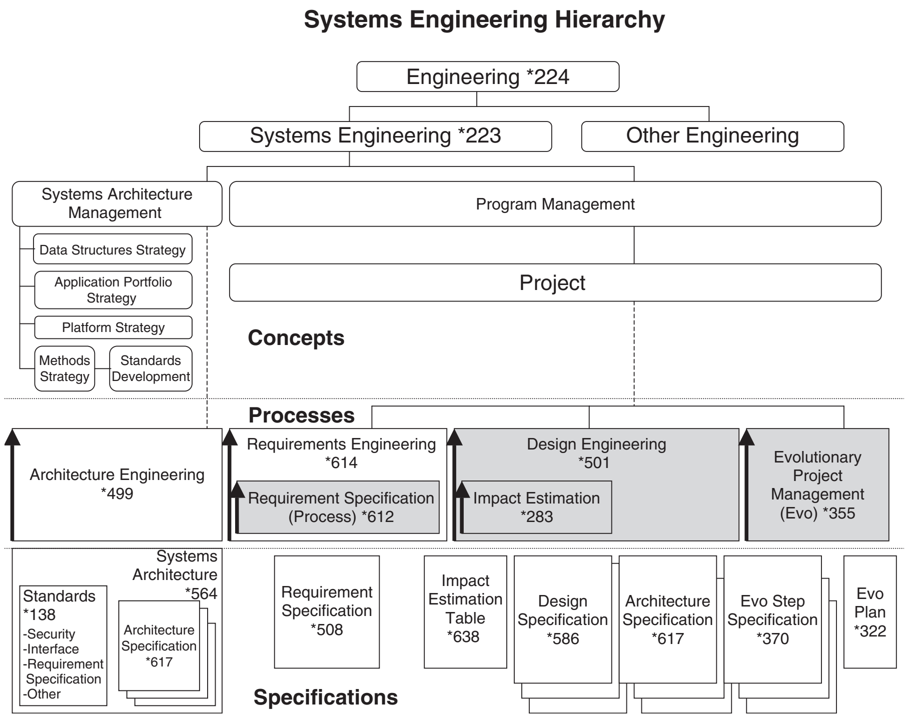

Systems Engineering Hierarchy
Engineering *224
Systems Engineering *223
Other Engineering
Systems Architecture Management
Program Management
Data Structures Strategy
Application Portfolio Strategy
Platform Strategy
Methods Strategy
Standards Development
Project
Concepts
Processes
Architecture Engineering *499
Requirements Engineering *614
Requirement Specification (Process) *612
Design Engineering *501
Impact Estimation *283
Evolutionary Project Management (Evo) *355
Systems Architecture *564
Standards *138
-Security
-Interface
-Requirement Specification
-Other
Architecture Specification *617
Requirement Specification *508
Impact Estimation Table *638
Design Specification *586
Architecture Specification *617
Evo Step Specification *370
Evo Plan *322
Specifications

Figure G30
Shows the relationship for systems engineering amongst concepts, processes and specifications.

2. Systems engineering includes a broad application of disciplines such as requirements engineering, quality control, project management, test engineering and any of the many other disciplines that might be found useful for satisfying stakeholders. Architecture Engineering, a subset of systems engineering, is by contrast, directed only towards the design aspects.

Synonyms: System Engineering *223.

Related Concepts: Engineering *224; Architecture Engineering *499; Requirement Engineering *614; Design Engineering *501; Evolutionary Project Management *355.

Tag **Concept *146**

A term that serves to identify a statement, or set of statements, unambiguously.

Notes:

1. A local tag name is declared for any set of specification statements which needs a 'unique identity' to enable local cross-referencing. It is a direct reference to the specific set of specification statements. For example, 'Local Tag 3.' A local tag must be unique within the specification it is declared in, without needing hierarchical tag references or any other device to locate it.
2. Hierarchical tags help people navigate, and understand the context of a specification. A hierarchical tag identifies a local tag name in some larger context. It can also be used to resolve ambiguity amongst two or more identical, local tags. For example, A.B.Button.XY and C.D. Button.XY. A hierarchical tag has a structure of Tag 1.Tag 2.Tag 3.....Local Tag N. Meaning Tag 3 is a subset of Tag 2. Tag 2 is a subset of Tag 1 – and that Local Tag N will be found in Tag 1's location.
   Hierarchical tags can have any useful number of levels, no matter how many levels are defined in total in a specification. You do not have to repeat the entire formal sequence of hierarchical tags – just specify enough to help find the local tag you are referring to, unambiguously.
3. One use for hierarchical tagging is to identify a specific Planguage statement. For example, Usability.Fail and Usability.Goal refer to the parameters under the Usability tag.
4. [Qualifier] conditions can be used to differentiate amongst several equal terms. The qualifier conditions act as an extension to the normal tag helping us to distinguish amongst different spaces within the scope of the same tag name.

   Example:
   Reliability [USA, Retail Dealers].
   Project Oasis.Requirements.Usability.Fail [USA, Retail Dealers].

Synonyms: Identifier *146; Tag Name *146.

Target **Concept *048**

A target is a specified stakeholder-valued requirement, which you are aiming to deliver under specified conditions. There are two kinds of target: 'scalar' and 'binary.' Scalar targets are specified using the parameters {Goal, Budget, Stretch, Wish}. A performance target is known as a 'goal' and a resource target is known as a 'budget.' Binary targets are function targets.

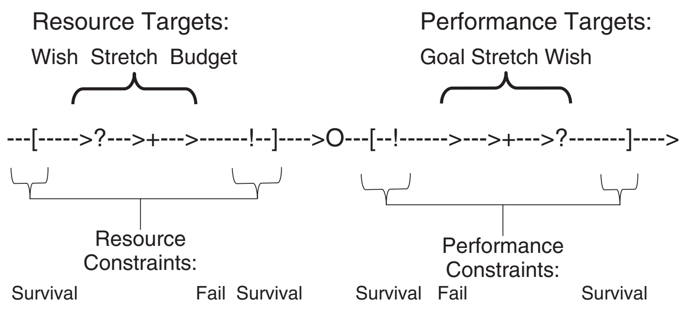


Figure G31
Shows scalar targets can be specified for both performance and resource requirements.

Notes:
1. A target is not a constraint. Targets specify the *success* levels for scalar requirements, and any stakeholder-desired binary requirements. Constraints by contrast specify any *failure* and/or *survival* levels for scalar requirements, and any *mandatory* requirements for binary requirements.
2. Function targets, valued functions, are subtly different from function constraints. Function constraints are mandatory. If a function constraint is not met, then some degree of failure will occur, or even total system catastrophe. Function targets do not have implied penalties, they are considered required by some stakeholder.
3. A scalar target specification consists of a numeric value (its target level) and its relevant [qualifiers]. As well as Goal, Budget, Stretch and Wish, 'Ideal' is a target parameter, but it is rarely used.
4. Target can also be used as a collective noun, applied to a set of function targets and variable scalar targets with their individual qualified levels.

Related Concepts: Goal *439: a performance target; Budget *421: a resource target; Benchmark *007; Constraint *218; Function Target *420.
Keyed Icon: @

Task **Concept *149**

A task is a defined and limited piece of work. A task may be defined formally by a procedure and other standards (how to carry out a task). A process is characterized by the repetition of a task.
Notes:
1. A task may be a complex activity. It can be defined using a set of process standards, such as procedures, rules, forms, rates, best practice models, checklists, and guidelines.

"A piece of assigned work."
<- *The American Heritage Dictionary.*

2. A task is the main part of a defined process. Entry conditions are checked before we invest time carrying out the task. We also check that a task has met our defined exit conditions before we consider ourselves properly done with it. This structure of a process is sometimes abbreviated as 'ETX' (Entry Task Exit).

Related Concepts: Act [PDSA] *172; Process *113; Procedure *115.

Test **Concept *256**

To test is to plan and execute an analytical process on any system, product or process, where we attempt to understand if the system performs as expected, or not.

A Test parameter can be used to reference specific test plans and processes.

Notes:

1. The overall aim of testing is to determine if the requirements are met.
2. Testing is a means of understanding *how* something works without necessarily understanding exactly why it works that way. Testing is from outside the 'black box.' Only by examining actual construction, or specification can we analyze the inner workings of the system (the design and construction).
3. We typically test by putting planned or random inputs into a system and comparing the resulting outputs (behavior and data) with our expectations. When the outputs deviate from expected behavior, we must analyze the reason to see if the system is failing to meet requirements or if the requirements are wrongly specified. If it is economic to do so, we will probably correct either the system itself, or the specification, or both.
4. The term testing *could* be applied in a much wider sense to *any* form of examination (QC, SQC, QA), but it is specifically limited in Planguage to 'input-output' testing of the system prior to operational use. 'Test' is not intended to apply to *conceptual* models of a system, but only to prototypes and real-world systems.
5. The Test parameter can also be used to specify, or more likely cross-reference, already-developed test cases and test plans for reuse or modification.

   Example:
   Requirement X:
   Scale: ....
   Meter [Module Level]: ..., [Customer Acceptance]: Contract Section 6.0, [Operation]: ...
   TP XYZ: Test Plan [System XYZ]: Test Plan Document XYZ [May This Year].
   Test [Goal]: TP XYZ [Section 1.2.3, Test Cases = {3.4, 6.7, 9.1.4}].
   Test parameter used to cross-reference test plans and test cases for Requirement X.
   Test Tag: Test [Acceptance]: Two independent observers, [Operation]: Built-in software test.
   A Test specification, which can be included with any attribute requirement, can be referred to via its tag ('Test Tag'). Notice that the qualifiers distinguish between two different stages of testing. The two suggested test methods are very roughly specified. This is useful at early stages of specification in order to get some idea and agreement about test costs and quality. This does not prevent a later stage of engineering detailing this to any interesting level of test plan.
6. Compare 'Test' with the parameter Meter, which is the specification of how to measure numerically according to a defined process in the Meter specification.

Related Concepts: Meter *093: Test differs from 'Meter' in that Test is very specifically concerned with system testing, while Meter is concerned with any form of measurement (for example, SQC).

Time **Concept *153**

Time defines '*when*'. It relates to any notion of time: clock-time or timescale. Time is used both as a parameter and in a qualifier condition.

Example:
Goal [Time = Weekends, Place = UK]: 60%.
Fail [Opening Hours, USA]: 50%.
Opening Hours: Time: {Weekday [0800 to 2000], Weekend [0900 to 1800], Not Legal Holiday}.
Goal [Time = By End of This Year]: 40%.
Goal [Date = Before January 31, This Year]: 30%.

Related Concepts: Place *107; Event *062; Qualifier *124; Scope *419.

Trend **Concept *155**

A Trend parameter is used to specify how we expect or estimate attribute levels to be in the future. It is used as a benchmark.

A Trend parameter states a numeric value, on a defined Scale, under specified conditions [time, place, event], for a scalar attribute that is extrapolated into the future, based on current knowledge.

Rationale: The purpose of Trend is to give us a better comparison (benchmark) for the degree of improvement or change we are planning or achieving, than we would get by using the more static benchmark 'Past.' Trend makes us plan to cope with the future, not just the past. It makes systems engineers think about the *competition*.

Example:
Peace:
Scale: Probability of Peacetime Situation.
Trend [Next Year, Country = {Gb, F, NO, DK}]: 80%? <- Marketing Guess.
Peacetime Situation: National Military Forces are not deployed anywhere for any purpose, even NATO or UN peacekeeping.
Trend represents our expectation of what the Past levels of this attribute will extrapolate to in the future (unless we plan to change that projected reality . . .).

"The other beauty ('truths I've learned to challenge') goes something like this: A team comes in with a proposal to leapfrog the current position of its leading competitor. The implicit assumption is the competition will be sleeping. Doesn't usually happen that way . . . It was tough, but we tried like hell to look at every new product plan in the context of what the smartest competitor could do to trump us. Never underestimate the other guy."

Jack Welch, former CEO General Electric (Welch 2001 Page 391)

Synonyms: Trend Level *155.
Related Concepts: Benchmark *007.
Keyed Icon: ?<
"Symbolizing a Past (<) with some doubt (?) about the perfect truth. Must normally be applied on a scalar arrow, <------?<-----O----?<---->"
Historical Note: This concept was suggested first by Kai Thomas Gilb, May 1995.

Type **Concept *398**

Type specifies the category of a Planguage concept. Categories for Type may be defined both by Planguage and by local extensions.

Notes:

1. Type classifications can be explicit.

   Example:
   Maintaining Standards:
   Type: Function. "Explicit specification of Type."
2. Type classifications can be implicit. Type can be implied by content and context.

   Example:
   Usability: Objective. "Implicit use of Type."
   Requirements Section. "Implicit Type given by use in a heading."
3. In this glossary, Type can be used explicitly to state the categories of the Planguage concepts (to save space, these have been omitted from this book).
4. Type can be specified as a hierarchy.

   Example:
   Requirements.Performance.Quality.
5. A specific specification type may *demand* certain rules of specification are followed, or *imply* certain properties. For example, a 'Performance' type will always require a defined scale of measure as it is scalar, but 'Design' will not, as it is binary.

Uncertainty **Concept *310**

Uncertainty is the degree to which we are in doubt about how an impact estimate, or measurement, of an attribute reflects reality. We are 'uncertain' as to whether the current or future reality is better or worse (than the observed or estimated value of an attribute), and by how much it differs.

Notes:

1. The reason behind the uncertainty could be either the expected, known variance in the results, or it could be the quality (accuracy, reliability, precision and relevance) of the measuring or estimating method, or both.
2. A 'risk' is a factor that could result in a future negative consequence. An uncertainty becomes a 'risk' when it implies a potential that a *future* result will be *negative* in relation to a planned or estimated target. (I am well aware of the field definitions used in Economics for risk and uncertainty (Bernstein 1996), and the history of making a distinction (for example, the work of Frank Knight and J. M. Keynes). However, the definitions here are tailored to system engineering purposes, rather than Economics.)

Related Concepts: Risk *309.

Until **Concept *551**

'Until' is a logical operator that is used to limit the extent of a scalar range of values. The purpose is to explicitly map a range rather than have it implied by a single value at one extreme (like a Fail limit).

Example:
Fail [Gb, Next Version]: 60% Until Survival.
Survival [International, Next Year]: 20% Until 0%.
Fail [EU]: 80% Until 20%.
Related Concepts: Or Worse *549; Or Better *550; Range *552.

User-Defined Term — Concept *530

In Planguage, a user-defined term is a definition of a term made by a Planguage user. It is not a Planguage term (like Scale or Goal), nor a customer-tailored Planguage term, such as a new parameter, parameter synonym or grammatical variation.
The scope of a user-defined term is 'local,' and it might apply within a specific definition, within a specific project or across an organization.
A user-defined term has a tag that it is referred to by. It may be specified using 'Defined' or 'Defined As' or, by adding a tag to give a tag name to any expression, statement, or term.

Example:
Address Change: Defined As: A change to an existing address.
MTBF: Scale: Mean Time Between Failure.
PB: Goal [If Peace]: 20%.
User-defined terms are Address Change, MTBF, PB, Failure and Peace. Address Change is an example of explicit definition. MTBF and PB are tags defined in the statements above. Failure and Peace are defined elsewhere.

Notes:
1. A user-defined term is not part of a Project Language (also known as a 'Specific Project Specification Language' – a customized Planguage Specification Language).

Synonyms: Project-Defined Term *530.
Related Concepts: Planguage Term *211; Project Language *247.

Value — Concept *269

Value is *perceived* benefit: that is, the benefit we think we will get from something.

Notes:
1. Value is the potential *consequence* of system attributes, for one or more stakeholders.
2. Value is not linearly related to a system improvement: for example, a small change in an attribute level could add immense perceived value for one group of stakeholders for relatively low cost.
3. Value is the *perceived* usefulness, worth, utility or importance of a defined system component or system state, for defined stakeholders, under specified conditions.

"One man's meat is another man's poison." *Old proverb*

4. 'Benefit' is when some perceived value is actually *produced by*, a defined system.
5. Value is relative to a stakeholder: it is not absolute. Quality, for example, is stated in terms of the objective level of 'how well' a system performs, irrespective of how this level is appreciated by any stakeholders. Some defined levels of quality only have a value to some

stakeholders. The same is true for all attributes. There are many Planguage ways of indicating that a stakeholder values an attribute. These include using Value, Stakeholder, Authority, Impacts, and Source parameters.

"Nowadays, people know the cost of everything and the value of nothing."
Oscar Wilde.

Synonyms: Worth *269.
Related Concepts: Benefit *009; Impacts *334; Values *592.

Version **Concept *332**

A version is an initial or changed specification instance. A version *identifier* can be made from any symbols. It is useful to indicate unique instances of a specification, also probably the *sequence* of changes, and perhaps even the exact *time* of change.
A version identifier is specified by the Version parameter. By default, use the date as the version identifier.
Notes:
1. The version should be specified at the level of individual elementary requirement and design specifications.
   Rationale: This aids change control. It allows reviewers to focus mainly on the changes themselves, rather than the entirety of large documents, which contain perhaps only a few changes. It also enables us to treat individual elementary specifications as relatively independent objects, which are electronically grouped as needed into useful views, rather than the traditional 'documents.'
   Example:
   Version: January 9, 2003.
   Edition: 1.02 [Feb 21 03 3:54:36 pm].
2. If a date alone is specified on the same line as a tag, and immediately after it, then that date will be understood as the version identifier for whatever that tag encompasses.
   Example:
   Usability: January 9, 2003. Scale: Time to <learn>. Goal: 6 hours or better.
   Usability: Version = January 9, 2003. Scale: Time to <learn>. Goal: 6 hours or better.
   Usability [Version = January 9, 2003]: Scale: Time to <learn>. Goal: 6 hours or better.

Synonyms: Edition *332; Instance *332.

Vision **Concept *422**

A vision is an idea about a future state, which is very long range and probably idealistic, maybe even unrealistic.
The future state is likely to be about the position of a corporation or product line in relation to the market – rather than about specific properties of a specific product or system.
Example:
"I say to you today, my friends, that in spite of the difficulties and frustrations of the moment I still have a dream. It is a dream deeply rooted in the American dream.

I have a dream that one day this nation will rise up and live out the true meaning of its creed — 'We hold these truths to be self evident, that all men are created equal'."

Martin Luther King Jr., Washington, DC[5]

Notes:

1. Top managers or leaders state a vision in order to create teamwork to move in a required direction. A vision can be analyzed and decomposed into a set of requirements.
2. In the speeches and writings of senior management, the vision might be the only defined component which is quoted. However, the organization and management would be wise to articulate and clarify their understanding of, and commitment to, the vision by decomposing it into a hierarchical set of specific objectives, including specific goals for the elementary objectives. They should map the path to the vision with both short-term and long-term *numeric* targets. They can then begin an evolutionary process of moving towards the specified vision.
3. A vision statement is the reference point for developing more-detailed specifications, such as product line performance specifications, that support the achievement of the vision. A vision statement presumes that at least an assumption is made about the mission, for example that 'we are in the mobile phone business,' or 'we make aircraft'.

Wish — Concept *244

A Wish parameter is used to specify a stakeholder-valued, uncommitted target level for a scalar attribute. A Wish level is specified on a defined Scale, under specified conditions [time, place, event]. There is no commitment to deliver a Wish level. A Wish parameter simply specifies some stakeholders' desired level, without considering its cost or practicality.

Notes:

1. Wish parameters can be useful for acknowledging and recording stakeholder desires (while clearly *not* committing to them) *until* suitable design ideas are identified, *until* resources are provided for those designs or perhaps *until* deadlines are adjusted.
2. Wish belongs to the set of target specifications: {Goal/Budget, Stretch, Wish}.
3. Subject to qualifying conditions, Wish specifications have the lowest scalar requirement priority. (The order from highest to lowest priority is Survival, Fail, Goal, Stretch and then Wish.)
4. A Wish specification can apply to a performance or resource requirement.

Rationale: If we did not have a Wish parameter to articulate uncommitted stakeholder needs, then this information might never be collected, and maintained. So, we might lose the competitive advantage of knowing what our stakeholders desire and value, when the resources or technology ultimately become available.

[5] From: http://www.ku.edu/carrie/docs/texts/mlkdream.html/. Martin Luther King, 'I Have a Dream' speech on August 28, 1963, Washington, DC.

Synonyms: Wish Target *244; Wish Level *244 (see Level *337).
Related Concepts: Need *599; Target *048; Goal *109; Budget *480; Stretch *404; Ideal *328.
Keyed Icon: >? "A perhaps questionable goal or budget. In context: ---->?--->O--->?---> "
Historical Note: The Wish parameter was first suggested in December 1995 by the Scottish Widows organization, through Dorothy Graham of Grove Consultants.

Workload Capacity **Concept *459**

Workload capacity is a performance attribute. It is used to express the capacity of a system to carry out its workload, that is 'how much' a system can do, did or will do.
Notes:
1. Workload capacity can be used to capture many different concepts of workload, such as maximum number of registered users, maximum number of concurrent users, maximum data volumes and average transaction response times.
2. Workload capacity expresses the system capability to perform a defined type of work.

Scale [Generic]: An amount of defined [Work Task] to be done in a defined [Time] by a defined [Agent] in a defined [Environment] on a defined [System].
Synonyms: Work Capacity *459; Workload Capability *459; Workload *459; Capacity *459.
Related Concepts: Performance *434; Quality *125; Resource Saving *429; Workload Capacity Requirement *544.

BIBLIOGRAPHY

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen and Juhani Warsta. 2002. *Agile Software Development Methods. Review and Analysis*. Espoo, Finland: VTT Publications. ISBN 951-38-6009-4. www.inf.vtt.fi/pdf/. 107 pages.

Akao, Yoji (Editor). 1990. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Cambridge, MA: Productivity Press Inc., ISBN 0-915-29941-0. 367 pages.

Ansoff, H. Igor. 1965. *Corporate Strategy*. London: Penguin Books. ISBN 0-140-09112-2. 288 pages.

Bernstein, L. 1996. *Against The Gods: The Remarkable Story of Risk*. Wiley. ISBN 0-471-29563-9. 383 pages.

Blanchard, Benjamin S. 1997. *System Engineering Management*. New York: John Wiley & Sons, Inc. ISBN 0-471-19086-1. 504 pages.

Bronson, Darren. 1999. Best Practices for Evolutionary Software Development. Massachusetts Institute of Technology (MIT) MBA and MSc. thesis. (This was submitted to the Sloan School of Management and the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Master of Business Administration and Master of Science in Electrical Engineering and Computer Science in conjunction with the Leaders for Manufacturing Program at the Massachusetts Institute of Technology in June 1999.

Cockburn, Alistair. 2002. *Agile Software Development*. Boston, MA: Addison Wesley, Inc. ISBN 0-201-69969-9. 278 pages.

Cotton, Todd. August 1996. Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion. *Hewlett-Packard Journal*. Volume 47, Number 4, Pages 25–38. (This is adapted from the book, *Object-Oriented Development at Work: Fusion in the Real World*. Ruth Malan et al. (eds). Englewood Cliffs, NJ: Prentice Hall PTR. 1996.)

Crosby, Philip B. 1985. *Quality Improvement through Defect Prevention: An Individual's Role*. Philip Crosby Associates Inc.

Crosby, Philip B. 1996. *Quality is Still Free: Making Quality Certain in Uncertain Times*. McGraw Hill. ISBN 0-07-014532-6. 265 pages.

Cusumano, Michael A. and Richard W. Selby. 1995. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. The Free Press (A Division of Simon and Schuster). ISBN 0-02-874048-3. 512 pages.

Daniels, J., P. W. Werner and A. T. Bahill, 2001. Quantitative Methods for Tradeoff Analysis. *Systems Engineering*. Volume 4, Number 3, Pages 190–211.

Delavigne, Kenneth and Daniel J. Robertson. 1994. *Deming's Profound Changes. When Will the Sleeping Giant Awaken?* Englewood Cliffs, NJ: Prentice Hall PTR. ISBN 0-13-292690-3. Foreword by W. E. Deming.

Deming, W. Edwards. 1986. *Out of the Crisis*. MIT Center for Advanced Engineering Study (CAES), Cambridge, MA, USA-02139. ISBN 0-911379-01-0. 507 pages.

Deming, W. Edwards. 1993. *The New Economics For Industry, Government, Education*. MIT Center for Advanced Engineering Study (CAES), Cambridge, MA, USA-02139. ISBN 0-911379-05-3. 240 pages.

Dion, Raymond. July 1993. Process Improvement and the Corporate Balance Sheet. *IEEE Software*. Pages 28–35. See Haley et al. (1995) for update and more detail.

DoD Evolutionary Acquisition. 1998. *Joint Logistics Commanders Guidance for Use of Evolutionary Acquisition Strategy to acquire Weapon Systems*. The Defense Systems Management College Press, Fort Belvoir, VA 22060-5565. Revised and reissued in June 1998 with a new Foreword.

Elliot, Trevor and Dave Herbert. 2002. *Joined-Up Systems: Building the Integrated Business (The Management Consultancies Association Series)*. London: Hodder and Stoughton Educational. ISBN 0-340-85054-X. 192 pages.

Gilb, Tom. 1976. *Software Metrics*. Sweden: Studentlitteratur AB, Lund (1976). USA: Winthrop (1977). Out of print.

Gilb, Tom. 1988. *Principles of Software Engineering Management*. Wokingham and Reading, MA: Addison-Wesley. ISBN 0-201-19246-2. 442 pages.

Gilb, Tom. 2005. Course lecture slides, articles, papers and book manuscripts on World Wide Web. URL: www.Gilb.com/.

Gilb, Tom and Dorothy Graham. 1993. *Software Inspection*. Addison-Wesley. ISBN 0-201-63181-4. 471 pages.

Grady, Robert B. and Tom Van Slack. Hewlett-Packard. July 1994. Key Lessons in Achieving Widespread Inspection Use. *IEEE Software*. Pages 46–58. This paper also appears in Wheeler, Brykczynski and Meeson (1996). This paper is available for a fee at http://csdl.computer.org/comp/mags/so/1994/04/s4046abs.htm/.

Haley, T., B. Ireland, Ed. Wojtaszek, D. Nash and R. Dion. Raytheon. 1995. Raytheon Electronic Systems Experience in Software Process Improvement. This paper is available on-line at http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.017.html/.

Hooks, Ivy F. and Kristin A. Farry. 2000. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. Amacom (www.amacombooks.org). ISBN 0-814-40568-1. 272 pages. See www.complianceautomation.com/.

Hummel, Helmut. September 2002. A Strategy for Acquiring Large and Complex Systems. Talk given by Dr. Helmut Hummel, NATO Conference, Bonn.

IBM Systems Journal (IBMSJ). 1980. The Management of Software Engineering. *IBM Systems Journal*. Volume 19, Number 4, Pages 414–477.

- Part 1: H.D. Mills. Principles of Software Engineering. See Mills (1980).
- Part IV: M. Dyer. Software Development Practices.
- Part V: R. E. Quinnan. Software Engineering Management.

Individual papers from the 1980 Journal are downloadable from http://www.research.ibm.com/journal/.

IBM Systems Journal (IBMSJ). 1990. *IBM Systems Journal*. Individual papers from the 1990 journal are downloadable from http://www.research.ibm.com/journal/.

IBM Systems Journal (IBMSJ). 1994. Software Quality. *IBM Systems Journal*. Volume 33, Number 1. Individual papers from the 1994 journal are downloadable from http://www.research.ibm.com/journal/.

Ireson, W. Grant. (Editor). 1966. *Reliability Handbook.* New York: McGraw Hill.

Jandourek, Emil. August 1996. A Model for Platform Development. *Hewlett-Packard Journal.* A copy of this article is available in PDF at http://www.hpl.hp.com/hpjournal/96aug/aug96a6.htm/.

Järkvik, Jack, Lars Kylberg and others. 1994. Om att Lyckas (A Bit of Luck Helps, Too: On Succeeding). Published by Ericsson, Kista, Sweden for internal use. Sponsored by Mr. Stellan Nennerfelt. Internal publication number EN/LZT 123 1987.

Juran, J. M. 1964. *Managerial Breakthrough: A New Concept of the Manager's Job and a Systematic Approach to Improving Management Performance.* McGraw Hill. ISBN 0-07-033172-3. 396 pages.

Juran (Editor). 1974. *Quality Control Handbook.* McGraw Hill. ISBN 0-07-033175-8. 1780 pages. After 1994, this book is retitled *Juran's Quality Handbook.*

Kaplan, Craig, Ralph Clark and Victor Tang. 1994. *Secrets of Software Quality, 40 Innovations from IBM.* McGraw Hill. ISBN 0-07-911975-3. 383 pages. www.iqco.com/.

Keeney, Ralph L. 1992. *Value-focused Thinking: A Path to Creative Decision-making.* Cambridge, MA and London: Harvard University Press. ISBN 0-674-93197-1. http://www.fuqua.duke.edu/faculty/alpha/keeney.htm/.

Kelly, J. 1990a. An Analysis of Defect Density found during Software Inspection. Proceedings of 15th Annual Software Engineering Workshop (NASA SEL-90-006), Jet Propulsion Labs., Pasadena, CA.

Kelly, J. C. 1990b. An Analysis of Jet Propulsion Laboratory's Two Year Experience with Software Inspections. Proceedings of the Minnowbrook Workshop on Software Engineering, Blue Lake, NY.

Koen, Billy V. 1984. Toward a Definition of the Engineering Method. *Proceedings of the ASEE-IEEE Frontiers in Education.* 14th Annual Conference, Philadelphia, PA. 3-5. October 1984. Pages 544–549. The paper also appeared in *Engineering Education.* December 1984. Pages 150–155. Also in Spring 1985 in *The Bent of Tau Beta Pi.* Pages 28–33. Reprinted there with permission from *Proceedings of the ASEE-IEEE Frontiers in Education.* A full page extract is in Gilb (1988). An extended and very interesting comment on the paper's ideas is in Koen (2003).

Koen, Billy Vaughn. January 2003. *Discussion of the Method: Conducting the Engineer's Approach to Problem Solving.* Oxford University Press. ISBN 0-195-15599-8. Pages 260. http://www.me.utexas.edu/faculty/people/koen.shtml/.

Larman, Craig and Victor Basili. June 2003. Iterative and Incremental Development: A Brief History. *IEEE Computer.* Pages 2–11. See www.craiglarman.com for a copy of this paper.

MacCormack, Alan. Winter 2001. Product-Development Practices that Work: How Internet Companies Build Software. *MIT Sloan Review.* Pages 75–84. amaccormack@hbs.edu.

Maier, Mark W. and Eberhardt Rechtin. 2002. *The Art of Systems Architecting.* 2nd Edition. ISBN 0-9493-0440-7. 303 pages.

May, Elaine L. and Barbara A. Zimmer. August 1996. The Evolutionary Development Model for Software. *Hewlett-Packard Journal.* Volume 47, Number 4, Pages 39–45. Available as pdf Adobe Acrobat file at http://www.hpl.hp.com/hpjournal/96aug/aug96a4.htm/.

Mays, Robert. June 1995. IBM Defect Prevention Process and Test. Slides in 12th International Conference and Expo on Testing Computer Software, Washington, DC. See his chapter on the Defect Prevention Process and his references in Gilb and Graham (1993).

Mills, H. D. 1980. The Management of Software Engineering. Part 1: Principles of Software Engineering. *IBM Systems Journal.* Volume 19, Number 4. Reprinted 1999 in *IBM Systems Journal.* Volume 38, Numbers 2 and 3. A copy is downloadable from http://www.research.ibm.com/journal/sj/194/ibmsj1904C.pdf/.

Mintzberg, Henry. 1994. *The Rise and Fall of Strategic Planning: Reconceiving Roles for Planning, Plans, Planners.* New York: The Free Press (A Division of Macmillian, Inc.). ISBN 0-02-921605-2. 458 pages.

Morris, Peter W. G. 1994. *The Management of Projects.* London: Thomas Telford. ISBN 0 7277 1693 X. 358 pages. The American Society of Civil Engineers. The website http://www.indeco.co.uk/ has additional recent papers by Professor Morris.

Muller, Gerrit. 1999. Requirements Capturing by the System Architect. Available via http://www.extra.research.philips.com/natlab/sysarch/RequirementsPaper.pdf/.

Pence, J. L. Pete and Samuel E. Hon III. Telecommunications Network Systems, Bellcore, Piscataway, NJ. October 1993. Building Software Quality into Telecommunications Network Systems. *Quality Progress.* Pages 95–97.

Peters, Tom. 1992. *Liberation Management, Necessary Disorganization for the Nanosecond Nineties.* New York: Knopf and London: Macmillan. ISBN 0-333-53340-2.

Ramo, Simon and Robin K. St.Clair. 1998. *The Systems Approach: Fresh Solutions to Complex Civil Problems through Combining Science and Practical Common Sense.* TRW, Inc., Manufactured in USA, KNI Incorporated, Anaheim, CA. 150 pages.

Royce, Walker. 1998. *Software Project Management: A Unified Framework.* Reading, MA: Addison Wesley Longman, Inc. ISBN 0-201-30958-0. 406 pages.

Saaty, Thomas L. 1988. Multicriteria Decision Making: The Analytical Hierarchy Process. *Decision Support Software.* McLean, VA.

Slater, Robert. 2000. *The GE Way Fieldbook: Jack Welch's Battle Plan for Corporate Revolution.* New York: McGraw Hill. ISBN 0-07-135481-6. 288 pages.

Sproles, Noel. University of South Australia. 2002. Formulating Measures of Effectiveness. *Systems Engineering.* Volume 5, Number 4, Pages 253–263. A copy of this paper is available online for a fee at http://www.interscience.wiley.com/.

Spuck, William J. December 1993. The Rapid Development Method (RDM). Jet Propulsion Lab (JPL). JPL D-9679 (internal document).

Thayer, Thomas A., Myron Lipow and Eldred C. Nelson. TRW. 1978. *Software Reliability: A Study of Large Project Reliability.* TRW Series on Software Technology. Vol. 2. Amsterdam: North-Holland. Out of print.

Upadhyayula, Sharma. January 2001. Rapid and Flexible Product Development: An Analysis of Evolutionary Software Projects at Hewlett Packard and Agilent. Massachusetts Institute of Technology (MIT) MSc. thesis. A copy of the thesis is available on the Resources web page at the site http://www.globalbusinessstrategies.com. (This was submitted to the System Design and Management Program in partial fulfillment of the requirements for the degree of Master of Science in Engineering and Management at the Massachusetts Institute of Technology in January 2001.)

Welch, Jack. 2001. *Jack: Straight from the Gut.* New York: Warner Business Books. ISBN 0-446-52838-2. 479 pages.

Weller, Edward. September 1993. Lessons from Three Years of Inspection Data. *IEEE* Software. efweller@stt.com

Wheeler, David A., Bill Brykczynski and Reginald N. Jr. Meeson. June 1996. *Software Inspection, An Industry Best Practice.* IEEE Computer Society Press. Order No. BP07340. ISBN 0-818-67340-0. 293 pages.

Young, Ralph R. 2001. *Effective Requirements Practices* (Addison Wesley Information Technology Series). Addison Wesley. ISBN 0-201-70912-0.

Further Reading

Boehm, B., C. Abts, A.W. Borwn, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer and B. Steece. 2000. *Software Cost Estimation with COCOMO II*. Prentice Hall. Updates are at http://sunset.usc.edu/research/cocomosuite/.

Burr, Adrian and Mal Owen. 1996. *Statistical Methods for Software Quality: Using Metrics for Process Improvement*. London: International Thompson Computer Press. ISBN 1-85032-171-X. 453 pages.

Dettmer, H. William. 1997. *Goldratt's Theory of Constraints: A Systems Approach to Continuous Improvement*. 1997. Milwaukee, WI: American Society for Quality Press. ISBN 0-87389-370-0. 378 pages.

Drucker, Peter F. 1992. *Managing for the Future: The 1990's and Beyond*. New York: Truman Talley Books/Dutton (Penguin Group). ISBN 0-525-93414-6.

Florac, William A., Robert E. Park and Anita D. Carleton. April 1997. *Practical Software Measurement: Measuring for Process Management and Improvement*. Software Engineering Institute (SEI). 240 pages. Reference: CMU/SEI-97-HB-003. A copy (Acrobat Reader) of this guidebook is downloadable from SEI web site http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97hb003.pdf/.

Gause, D. C. and G. M. Weinberg. 1989. *Exploring Requirements: Quality Before Design*. New York: Dorset House Publishing Co. ISBN 0-932-633137. 320 pages.

Gilb, Tom. January 1996. What is Level 6? *IEEE Software*. Managers Column.

Gilb, Tom. 1997. *Evolutionary Project Management*. Unpublished draft book manuscript available at www.Gilb.com/.

Gilb, Tom. June 1997. Requirements-Driven Management: A Planning Language. *Crosstalk*. Pages 18–42. This paper is available on the DoD's Software Technology Support Center (STSC) website at http://www.stsc.hill.af.mil/Crosstalk via 'Back Issues' selecting June 1997.

Gilb, Tom. December 1998. Impact Estimation Tables: Understanding Complex Technology Quantitatively. *Crosstalk*. This article can be found on the DoD's Software Technology Support Center (STSC) website at http://www.stsc.hill.af.mil/CrossTalk via 'Back Issues' selecting December 1998.

Gilb, Tom. June 1999. Optimizing Systems Engineering Specification Quality Control Processes. *Proceedings: INCOSE, Brighton, UK*. Also published as Optimizing Software Inspections. March 1998. *Crosstalk*. This paper is available at http://www.stsc.hill.af.mil/CrossTalk via 'Back Issues' selecting March 1998.

Grady, Robert B. August 1996. Software Failure Analysis for High Return Process Improvement Decisions. *Hewlett-Packard Journal*. Volume 47, Number 4, Pages 15–24. Available as Adobe Acrobat pdf file at http://www.hpl.hp.com/hpjournal/96aug/aug96a2.htm/.

Gregory, Robin and Ralph L. Keeney. August 1994. Creating Policy Alternatives Using Stakeholder Values. *Management Science*. Volume 40, Number 8, Pages 1035–1048.

Hughes, Thomas P. 1998. *Rescuing Prometheus: Four Monumental Projects that Changed the Modern World*. Vintage (A division of Random House, www.vintagebooks.com). ISBN 0-679-73938-6. 372 pages.

Humphrey, W. S. 1989. *Managing the Software Process (SEI)*. Addison-Wesley. ISBN 0-201-18095-2. 450 pages. Humphrey's work on the Personal Software Process and Team Software Process is described at http://www.sei.cmu.edu/tsp/.

Kaplan, Robert S. and David P. Norton. 1996. *The Balanced Scorecard: Translating Strategy into Action*. HBS Press. ISBN 0-87584-651-3. 322 pages.

Kauffman, Stuart. 1995. *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*. Oxford University Press (OUP) Australia and New Zealand. ISBN 0-195-09599-5. 321 pages.

Kearns, David T., and David A. Nadler. 1993. *Prophets in the Dark, How Xerox Reinvented Itself and Beat Back the Japanese*. New York: HarperBusiness. (Permissions: HarperCollins Publishers, 10 E 53rd, NY NY 10022 Fax (212) 207-7222) ISBN 0-887-30564-4. 320 pages.

Keeney, Ralph L. Summer 1994. Creativity in Decisionmaking with Value-focused Thinking. *Sloan Management Review*. Volume 35, Number 4, Pages 33–41. Reprint No. 3543.

Keeney, Ralph L. September-October 1994. Using Values in Operations Research. *Operations Research*. Volume 42, Number 5, Pages 793–813.

Larman, Craig. 2003. *Agile and Iterative Development: A Manager's Guide*. Addison Wesley. ISBN 0-131-11155-8. 320 pages. See www.craiglarman.com for sample chapters.

MacCormack, Alan (Harvard University), Chris F. Kemerer (University of Pittsburgh), Michael Cusumano (MIT) and Bill Crandall (Hewlett-Packard). September/October 2003 Exploring Trade-offs between Productivity and Quality in the Selection of Software Development Practices. *IEEE Software*. Volume 20, Number 5, Pages 78–85.

Marshall, Lisa J. and Lucy D. Freedman. 1995. *Smart Work: The Syntax Guide for Mutual Understanding in the Workplace*. Kendall/Hunt Publishing Company, 4050 Westmark Drive, Dubuque, Iowa 52002. ISBN 0-7872-0491-9. 158 Pages.

Mays R. G., C. L. Jones, G. J. Holloway and D. P. Studinski. 1990. Experiences with Defect Prevention. *IBM Systems Journal*. Volume 29, Number 1, Pages 4–32. A copy of this paper is available at http://www.research.ibm.com/journal/.

Norman, Donald A. 1988. *The Design of Everyday Things*. Basic Books (September 2002, Paperback). ISBN 0-465-06710-7. Earlier editions are out of print.

Persico Jr., John and Gary N. McLean. April 1994. Manage With Valid Rather Than Invalid Goals. *Quality Progress*. Pages 49–53.

Peters, Tom. 1994. *The Tom Peters Seminar: Crazy Times Call for Crazy Organizations*. Vintage (Random House). ISBN 0-679-75493-8.

Peters, Tom. 2000. *Reinventing Work, the Project 50*. New York: Alfred A. Knopf. ISBN 0-375-40773-1.

Petroski, Henry. 1992. *The Evolution of Useful Things*. New York: Vintage Books (A division of Random House, Inc.). ISBN 0-679-74039-2, 288 pages.

Robbins, Harvey and Michael Finley. 1997. *Why Teams Don't Work: What Went Wrong and How to Make It Right*. London: Orion Business. ISBN 0-75281-350-1.

Saaty, T. L. 1980. *The Analytical Hierarchy Process*. New York: McGraw Hill.

Sorensen, Reed. Software Technology Support Center. January 1995. A Comparison of Software Development Methodologies. *Crosstalk*. Pages 12–18. A copy of this paper is available at http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri*1/4*1995/01/Comparis.asp.

Strauss, S. and Ebenau, R. Copyright by AT&T. 1994. *Software Inspection Process*. McGraw Hill. ISBN 0-07-062166-7. 363 Pages. First available November 1993.

Weber, Matthias and Joachim Weisbrod. DaimlerChrysler Research RIC/SM and SP. January/February 2003. Requirements Engineering in Automotive Development – Experiences and Challenges. *IEEE Software*. Volume 20, Number 1, Pages 16–24.

Weinberg, G. M. 1991. *Quality Software Management: Systems Thinking (Volume 1)*. New York: Dorset House. ISBN 0-932-63322-6. 336 pages.

Weinberg, G. M. 1992. *Quality Software Management: First-Order Measurement (Volume 2)*. New York: Dorset House. ISBN 0-932-63324-2. 360 pages.

Weinberg, G. M. 1994. *Quality Software Management: Congruent Action (Volume 3)*. New York: Dorset House. ISBN 0-932-63328-5. 328 pages.

Weinberg, G. M. 1997. *Quality Software Management: Anticipating Change (Volume 4)*. New York: Dorset House. ISBN 0-932-63332-3. 480 pages.

Concept Index

Concept Number	Concept Name	Page
*033	**Cost**	344
*033 Syn	**Expense**	344
*033 Syn	**Price**	344
*034	**Price of Nonconformance**	See website
*035	**Credibility**	345
*036	**Critical Factor**	345
*038	**Consumer**	See website
*038 Syn	**Customer**	See website
*041	**Defect Detection Process**	345
*041 Acron	**DDP**	345
*042	**Defect Prevention Process**	346
*042 Acron	**DPP**	352
*043	**Specification Defect**	See Error *274 & website
*043 Abbr	**Defect**	See Error *274 & website
*044	**Definition**	346
*044 Syn	**Defined**	347
*044 Syn	**Defined As**	347
*045	**And**	325
*046	**Design Process**	351
*047	**Design Idea**	350
*047 Syn	**Design**	351
*047 Syn	**Means**	351
*047 Syn	**Proposed Solution**	351
*047 Syn	**Strategy**	351
*048	**Target**	430
*048 Syn	**Target Level**	438
*049	**Delivery Cycle**	See Result Cycle *122 & website
*049 Syn	**Delivery**	See Result Cycle *122 & website
*051	**Specification Quality Control**	419
*051 Acron	**SQC**	420
*051 Syn	**Inspection**	371
*051 Syn	**Peer Review**	420
*052	**Downstream**	See website
*053	**Effectiveness**	See website
*054	**Efficiency**	See website
*054 Syn	**Cost Effectiveness**	See website
*055	**Elementary**	353
*056	**Entry Condition**	See website & Process *113
*057	**Entry Process**	See website & Process *113
*057 Abbr	**E**	See website & Process *113
*058	**Estimate**	354
*059	**Estimate, To**	354
*060	**Remaining Major Defects/Page**	See website & SQC *051
*061	**Evaluation Effort [SQC]**	See website & SQC *051
*062	**Event**	354
*063	**Evidence**	355
*064	**Exit Condition**	See website & Process *113

Concept Number	Concept Name	Page
*065	**Exit Process**	See website & Process *113
*065 Abbr	**X**	See website & Process *113
*067	**Fix Probability [SQC]**	See website & SQC *051
*068	**Form**	See website
*069	**Function**	360
*074	**Function Requirement**	363
*080	**Fuzzy**	365
*081	**Generic**	See website
*082	**High-Level**	See website
*083	**Hierarchical**	See website
*085	**Drawn Icon**	See website
*086	**Idea**	See website
*087	**Impact**	367
*088	**Process Improvement Suggestion**	See website & SQC *051
*089	**Logging [SQC]**	See SQC *051 & website
*091	**Major Defect**	374
*091 Abbr	**M**	375
*091 Abbr	**Major**	375
*093	**Meter**	376
*095	**Metric**	376
*096	**Minor Defect**	377
*096 Abbr	**m**	377
*096 Abbr	**Minor Defect**	377
*097	**Mission**	377
*098	**Fail**	359
*098 Syn	**Fail Level**	360
*098 Syn	**Fail Limit**	360
*098 Syn	**Failure**	360
*098 Syn	**Warning**	360
*098 Syn History	**Must (Avoid)**	360
*099	**Object**	See website & *145
*099 Syn	**System**	See website & *145
*100	**Objective**	379
*100	**Performance Requirement**	384
*101	**Open-ended**	See website
*102	**Owner**	380
*103	**Logical Page**	374
*103 Abbr	**Page**	374
*103 Acron	**LP**	374
*105	**Parameter**	380
*105 Syn	**Specification Language Parameter**	380
*106	**Past**	381
*106 Syn	**Past Level**	381
*107	**Place**	387
*108	**Plan**	See website
*109	**Goal**	366
*109 Syn	**Goal Level**	367
*109 Syn	**Planned Goal**	367
*109 Syn History	**Plan**	367

Concept Number	Concept Name	Page
*190	**Designer**	See website
*190 Syn	**Systems Designer**	See website
*192	**Architecture**	325
*193	**Systems Architect**	See website
*194	**Interface**	See website
*196	**Evolutionary**	358
*196 Abbr	**Evo**	358
*197	**Review**	408
*198	**Scalar**	412
*199	**Resource**	403
*208	**Principle**	See website
*211	**Planguage Term**	See website & Planguage *030
*218	**Constraint**	341
*220	**Baseline To Target Pair**	See website & IE *283
*222	**Subset**	See website
*223	**Systems Engineering**	428
*224	**Engineering**	See website & Systems Engineering *223
*233	**Stakeholder**	420
*234	**User**	See website
*235	**Client**	See website
*235 Syn	**Customer**	See website
*237	**Decision-maker**	See website
*237 Syn	**Decision-taker**	See website
*239	**Specify, To**	See website
*244	**Wish**	437
*244 Syn	**Wish Level**	438
*244 Syn	**Wish Target**	438
*249	**Binary**	334
*253	**Role**	409
*254	**Template**	See website
*256	**Test**	432
*259	**Rationale**	398
*259 Syn	**Justification**	399
*259 Syn	**Reason**	399
*262	**Systemic**	See website
*263	**Root Causes**	See website
*264	**Supra**	See website
*265	**Sibling**	See website
*266	**Kid**	See website
*267	**Parent**	See website
*269	**Value**	435
*269 Syn	**Worth**	436
*270	**Problem**	See website
*272	**Credibility Rating**	See website & Credibility *035
*273	**Side Effect**	417
*274	**Error**	353
*275	**Malfunction**	See Error *274 & website

Concept Number	Concept Name	Page
*276	**Issue**	372
*277	**Editor**	See website
*279	**Quality Control**	See website
*279 Acron	**QC**	See website
*281	**Team Leader**	See website & SQC *051
*281 Syn	**Moderator**	See website & SQC *051
*281 Syn	**SQC Leader**	See website & SQC *051
*281 Syn	**SQC Team Leader**	See website & SQC *051
*282	**Quality Assurance**	See website
*282 Acron	**QA**	See website
*283	**Impact Estimation**	368
*283 Acron	**IE**	368
*284	**Planning [SQC]**	See SQC *051 & website
*285	**Entry [SQC]**	See SQC *051 & website
*285 Syn	**SQC Entry**	See SQC *051 & website
*286	**Kickoff [SQC]**	See SQC *051 & website
*287	**Checking [SQC]**	See SQC *051 & website
*288	**Edit [SQC]**	See SQC *051 & website
*289	**Edit Audit [SQC]**	See SQC *051 & website
*289 Syn	**Follow Up**	See SQC *051 & website
*290	**Exit [SQC]**	See SQC *051 & website
*290 Syn	**SQC Exit**	See SQC *051 & website
*291	**Upstream**	See website
*293	**Debugging**	See website
*294	**Non-Commentary**	378
*295	**Readership**	399
*295 Syn	**Intended Readership**	399
*296	**Editor Advice Log**	See website & SQC *051
*296 Syn	**Author Advice Log**	See website & SQC *051
*296 Syn	**Issue Log**	See website & SQC *051
*297	**Report [SQC]**	See SQC *051 & website
*298	**Sample**	See website & SQC *051
*299	**Chunk**	See website & SQC *051
*300	**Item [SQC]**	See website & SQC *051
*301	**Question Of Intent**	See website & SQC *051
*302	**Expression**	See website
*303	**Master Definition**	375
*306	**Percentage Impact**	381
*306 Syn	**%Impact**	381
*306 Syn	**Incremental Percentage Impact**	381
*306 Syn	**Percentage Incremental Impact**	381
*307	**Incremental Scale Impact**	381
*308	**Cell**	See website & IE *283
*309	**Risk**	409
*309 Syn	**Threat**	409
*310	**Uncertainty**	434
*311	**Waterfall Method**	See website
*312	**Before**	333
*313	**After**	324

Concept Number	Concept Name	Page
*314	**During**	352
*318	**Incremental Development**	369
*319	**Data**	See website
*320	**Information**	See website
*321	**Verification**	See website
*322	**Evo Plan**	356
*322 Syn	**Evolutionary Delivery Plan**	356
*322 Syn	**Evolutionary Plan**	356
*328	**Ideal Historical**	See website
*331	**Input**	See website
*331 Syn	**System Input**	See website
*332	**Version**	436
*332 Syn	**Edition**	436
*332 Syn	**Instance**	436
*333	**Rule**	409
*334	**Impacts**	369
*337	**Level**	374
*337 Syn	**Point**	374
*339	**Fault**	See Error *274 & website
*339 Syn	**Bug**	See Error *274 & website
*339 Syn	**System Defect**	See Error *274 & website
*340	**Sum for Requirement**	See website & IE *283
*340 Syn	**Sum Requirement**	See website & IE *283
*342	**Backroom**	331
*343	**Frontroom**	360
*344	**Funder**	See website
*349	**Value To Cost Score**	See website
*351	**Baseline**	331
*353	**Kin**	373
*355	**Evolutionary Project Management**	358
*355 Abbr	**Evo**	355
*355 Acron	**RDM**	358
*355 Syn	**Evo Management**	358
*355 Syn	**Evolutionary Delivery Management**	358
*355 Syn	**Milestone Approach**	358
*355 Syn	**Rapid Delivery Management**	358
*355 Syn	**Result Delivery**	358
*355 Syn	**Synch-and-Stabilize**	358
*356	**Acquisition**	See website
*357	**Integration**	See website
*358	**Deployment**	See website
*359	**Gap**	365
*359 Syn	**Residue = Very Strange Term!**	365
*360	**Quality Level**	396
*360 Syn	**Major Defect Density**	396
*360 Syn	**Specification Quality Level**	396
*363	**Parallel Development**	See website
*364	**Serial Development**	See website
*370	**Evo Step Specification**	See website & Evo *355

Concept Number	Concept Name	Page
*419	Scope	416
*420	Function Target	364
*422	Vision	436
*423	Ambition	324
*424	Continuous Process Improvement	344
*424 Acron	CPI	344
*429	Resource Saving	406
*429 Syn	Saving	406
*431	Resource Requirement	405
*433	Impact Estimate	367
*433 Syn	Impact	367
*434	Performance	382
*436	Resource Target	406
*436 Syn	budget See also Budget *480	406
*438	Performance Constraint	383
*439	Performance Target	386
*439 Syn	goal See also Goal *109	386
*440	Survival	424
*440 Syn	Survival Level	425
*440 Syn	Survival Limit	425
*446	Scale Variable	415
*447	Default	See website
*447 Syn	Default Option	See website
*447 Syn	Default Value	See website
*453	Quality Requirement	See website
*453 Syn	Objective	See *100
*456	Specification Variable	See website
*456 Syn	Variable	See website
*459	Workload Capacity	438
*459 Syn	Capacity	438
*459 Syn	Work Capacity	438
*459 Syn	Workload	438
*459 Syn	Workload Capability	438
*460	Guideline	See website
*460 Syn	Advice	See website
*460 Syn	Hint	See website
*460 Syn	Recommendation	See website
*460 Syn	Suggestion	See website
*460 Syn	Tip	See website
*465	Dynamic	See website
*466	Statistical Process Control	See website
*466 Acron	SPC	See website
*469	Function Constraint	361
*472	Design To Cost	See website
*475	Deviation	352
*475 Syn	Variance	352
*478	Resource Constraint	404
*480	Budget – See also Resource Target *436	334
*480 Syn	Budget Level	335

AUTHOR INDEX

Subject Index