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

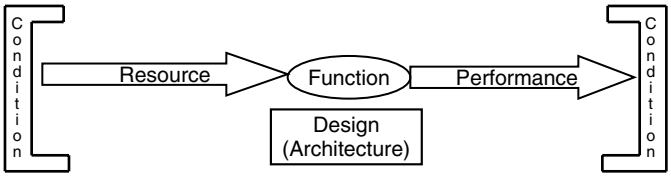


Figure 2.1
The basic system attributes describing a system.

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 around end 1970s,
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.cs.umb.edu/jfklibrary/j052561.htm, 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.*)

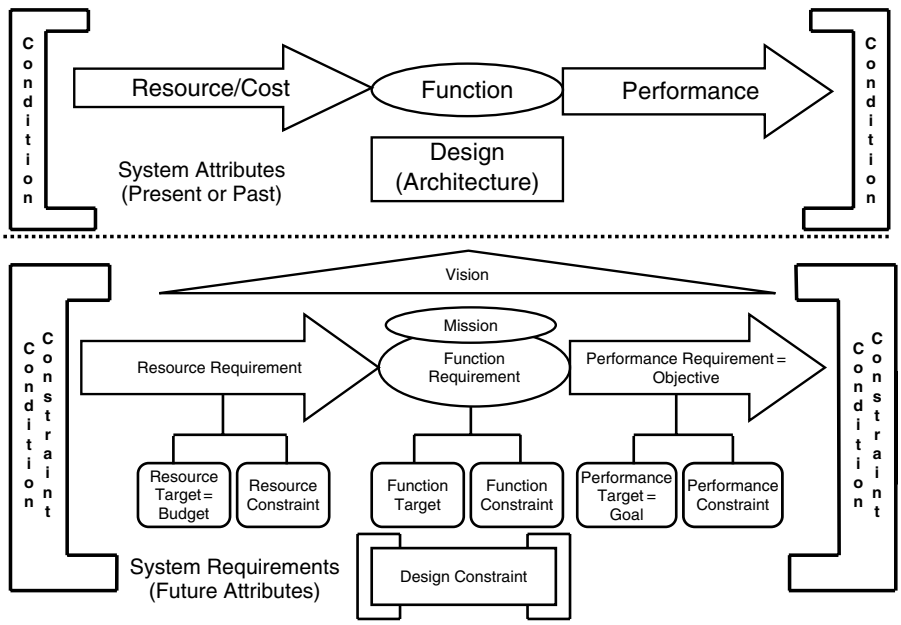


Figure 2.2
Mapping of system attributes to requirements.

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

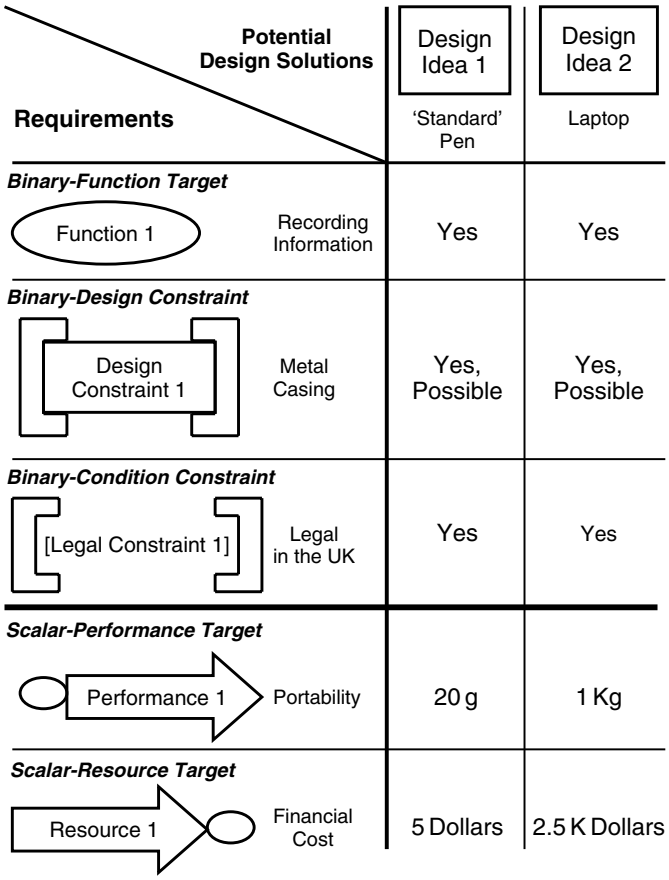
Euro Safety Design [European Models]:
Type: Design Constraint.
Description:
Use designs {X, Y, Z},
Do not use designs {M, N, P}.
Authority: European Safety Law.
Responsible Manager: Corporate Safety Director.
Implementer: Product Line Architect.

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.



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 life-cycle. 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

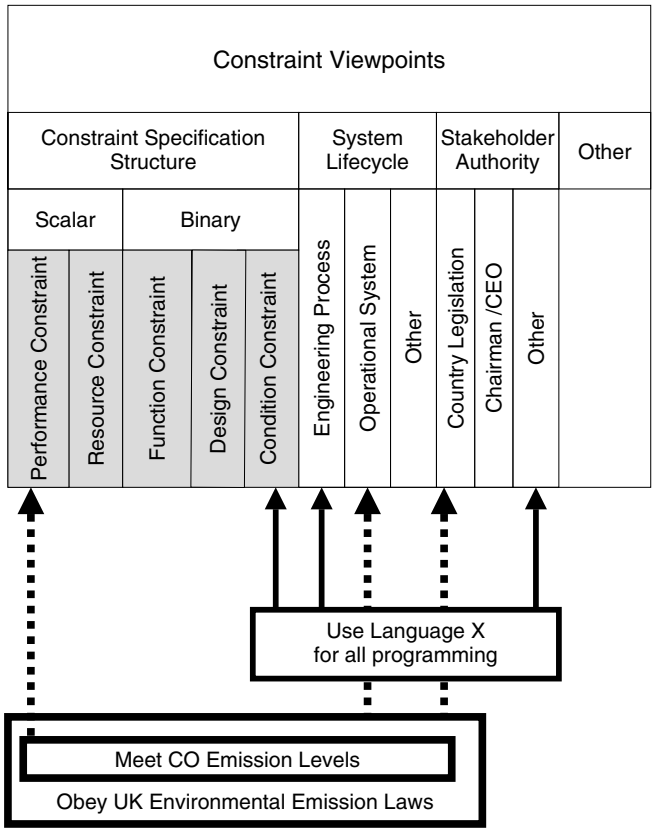


Figure 2.4
Constraint viewpoints: constraints are either scalar or binary. They can be categorized from several viewpoints.

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.

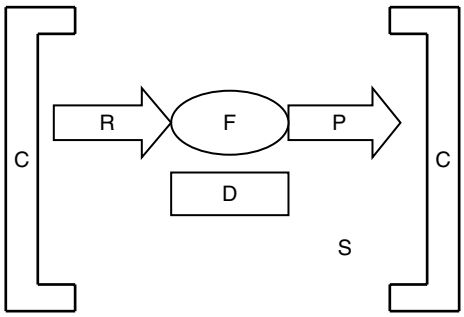
Bill of Rights

- 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)

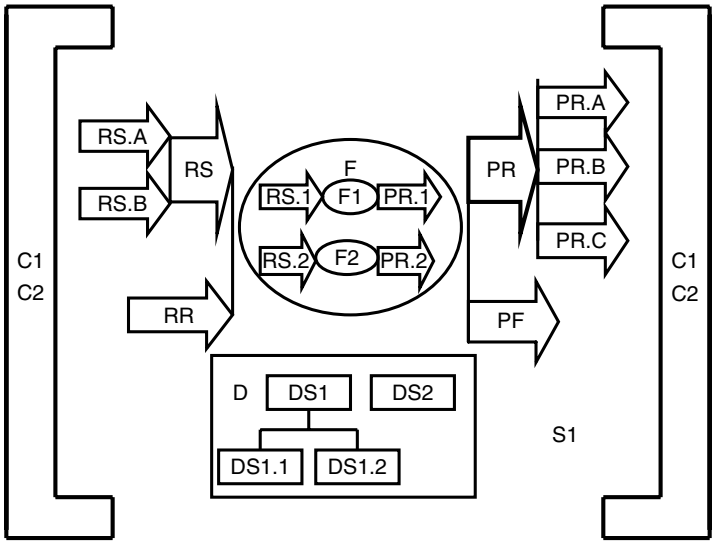
Figure 2.5
The author suggested these 'rights' for a multinational client. Of course it is a sneaky way to tell people what their 'duties' are!

2.9 Diagrams/Icons: Requirement Specification



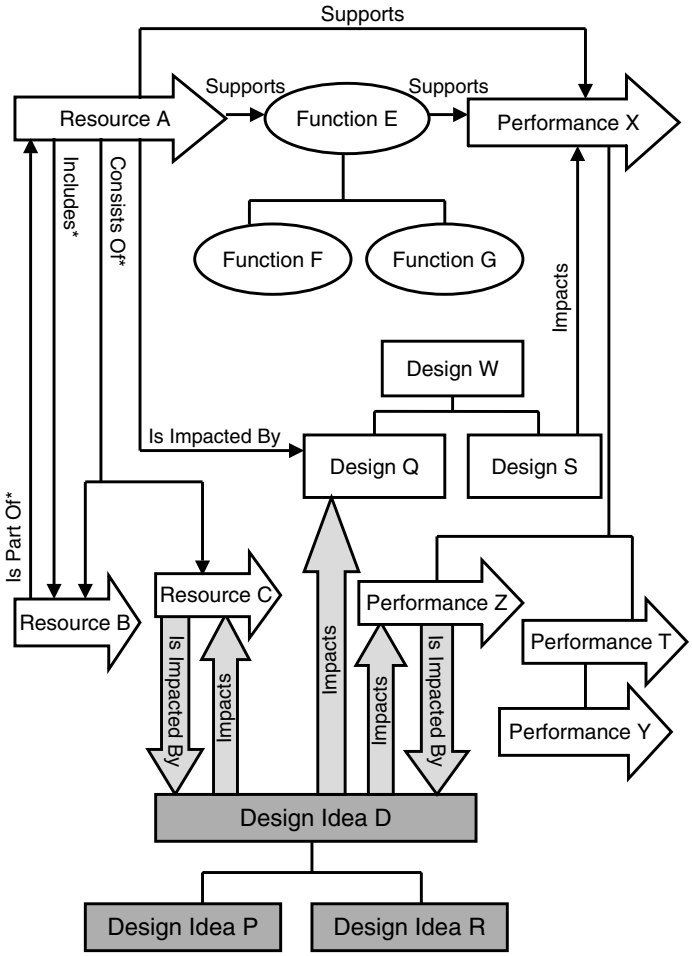
F is a function attribute. R is resource attribute, shown as an input attribute to F.
P is a performance attribute, shown as an output attribute to F.
D is a design attribute.
C represents a condition attribute (the two 'brackets' combined).
The set {R, F, P, D, C} make up a system, S.

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.

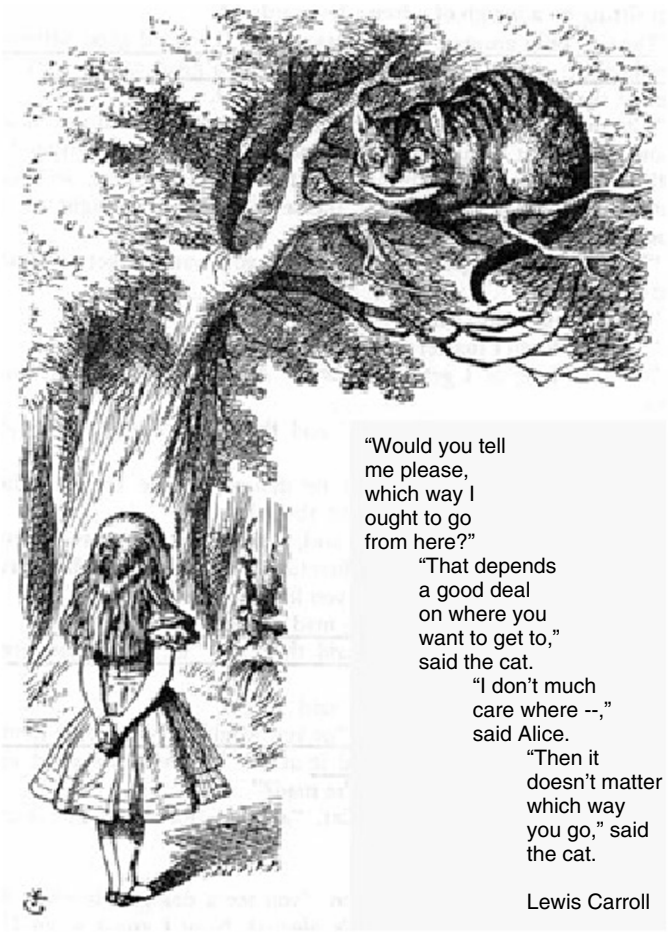


Figure 2.10
Alice and the Cheshire Cat. Illustration by John Tenniel, wood-engraving by Thomas Dalziel. From Chapter 6, *Alice in Wonderland* by Lewis Carroll.