2 A Generic Process for Requirements Engineering

If you can't describe what you are doing as a process, you don't know what you're doing.

William Edwards Deming, management consultant, 1900–93

2.1 Introduction

This chapter introduces the concept of a process for the development of systems. It starts by examining the way in which systems are developed. This leads to the identification of a development pattern that can be used in many different contexts. This development pattern is expressed as a generic process and is explained in some detail. Subsequent chapters indicate how the generic process can be instantiated for specific purposes. The relationship between process models and information models is also explored and an information model for the generic process is developed.

2.2 Developing Systems

Before any system can be developed it is essential to establish the need for the system. If the purpose of a system is not known, it is unclear what sort of system will be developed, and it is impossible to determine whether the system, when developed, will satisfy the needs of its users. Forest Gump summed it up nicely when he said:

If you don't know where you are going, you are unlikely to end up there.

The rigour with which the need is expressed will depend on the nature of the individual responsible for stating the need and their role within the organization in which they work. The need may be expressed in fairly vague terms initially, for example, "I would like a system that improves the efficiency of my department". Clearly, such a "specification" is not appropriate to be used as the basis for going out to buy a system. However, it could be the basis for a study to determine exactly what the person really wants. Such a study would have to determine where the department is currently inefficient and to postulate how the capabilities to be provided by the proposed system would be used to improve the efficiency. These activities, which transform a vague statement of need into a set of requirements that can be used as the basis for purchasing a system, constitute the process of developing the stakeholder requirements. Stakeholders include people who will directly interact with the system, but also other people and organizations that have other interests in its existence. The topic of creating stakeholder requirements is dealt with in detail in Chapter 5.

Figure 2.1 illustrates the development process. In the diagrammatic conventions used for process models, circles (or ovals) represent processes and rectangles represent data or information that is read or produced. The arrows indicate whether data is read or written. Thus, Figure 2.1 states that the *develop stakeholder requirements* process takes the *statement of needs* and produces the *stakeholder requirements*. It also creates and reads a *use model*.

Once a sound set of stakeholder requirements exist that define what the stakeholders want to be able to do with the proposed system, it is possible to begin to think about potential solutions. Rather than jumping straight to a design, it is good practice first to determine what characteristics the system must have irrespective of the final detailed design. This process is known as establishing the system requirements. It is recommended that an abstract model of the proposed system be produced. This model provides a basis for discussion within the development team and hence provides a means of establishing a common understanding

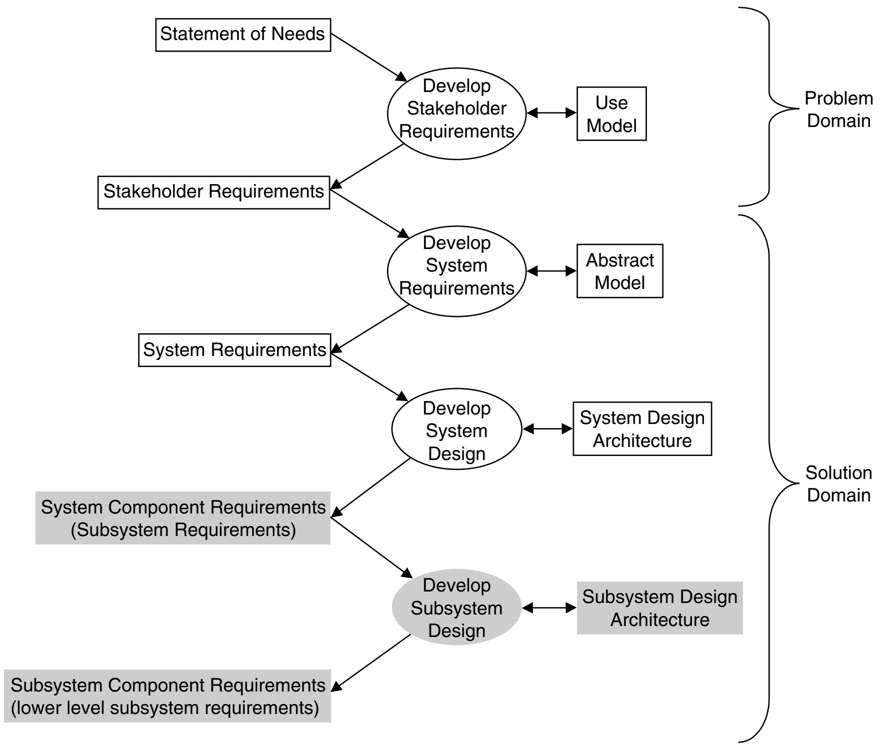

Statement of Needs
Develop Stakeholder Requirements
Use Model
Stakeholder Requirements
Develop System Requirements
Abstract Model
System Requirements
Develop System Design
System Design Architecture
System Component Requirements (Subsystem Requirements)
Develop Subsystem Design
Subsystem Design Architecture
Subsystem Component Requirements (lower level subsystem requirements)
Problem Domain
Solution Domain

Figure 2.1 System development process.

of the proposed solution, albeit at an abstract level. The model can also be used to explain the solution concepts to those stakeholders who wish to be assured that the developers are moving along the right lines. Finally, the model provides a structure for presenting the system requirements in a document form. Each element in the model can form a section in the document. This places each requirement in a relevant context and is an indispensable aid to reviewing the complete requirements set from a consistency and completeness point of view.

From the system requirements it is possible to consider alternative design architectures. A design architecture is expressed as a set of interacting components that collectively exhibit the desired properties. These properties are known as the emergent properties of the system and should exactly match the desired characteristics of the system as expressed in the system requirements. The design architecture defines what each system component must do and how the system components interact with each other to produce the overall effects specified in the system requirements. In other words, the design architecture defines the requirements for each system component (see Figure 2.1) in terms of their functionality and interaction obligations. The design architecture and hence the system component requirements must also stipulate any other required properties such as physical size, performance, reliability and maintainability.

For all but the smallest of systems, the components in the design architecture will be too complex to be implemented directly. Components at this level are frequently known as "subsystems" because they are complex enough to be considered as systems in their own right, but yet they are still only part of the higher level system for which they are designed.

The process of establishing the design architecture for each subsystem and then using this to derive component requirements is similar to that described for the overall system. Eventually a subsystem design architecture and subsystem component requirements will be produced for each subsystem as indicated in Figure 2.1.

This description of the development process has indicated that development of systems takes place at several levels and that different activities take place at each level. Figure 2.1 also indicates that each activity is supported by a model (*e.g.* use model, abstract model, design architecture), although the nature of the models differs significantly. This is an example of a common aspect: each level of development uses a model. In the following sections of this chapter, these similarities are further explored in order to define the properties of a generic process.

It is essential to realize that there are requirements at each of the levels:

- needs statement;
- stakeholder requirements;
- system requirements;
- system component requirements;
- subsystem component requirements.

Consequently, requirements engineering is not something that is done once and then forgotten. It happens at each level, and often work at different levels is undertaken concurrently. At all levels from the system components downward, there is multiple concurrent work on requirements at each level. (The grey background of the relevant symbols in Figure 2.1 indicates this.)

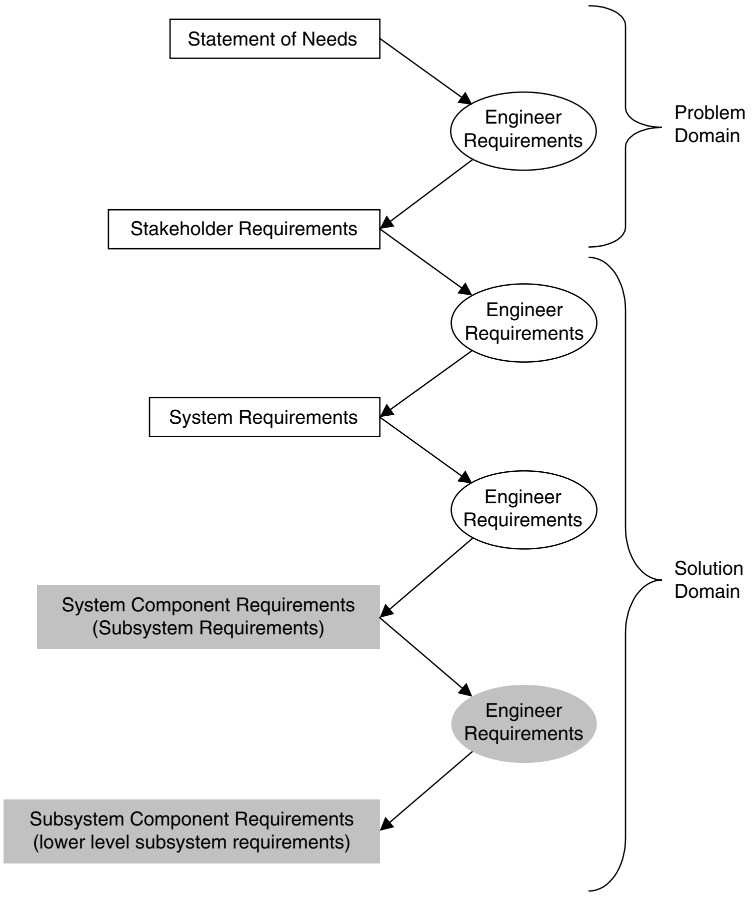


Figure 2.2 Different levels of requirements engineering.

2.3 Generic Process Context

An alternative way of considering the development process is shown in Figure 2.2. This diagram suggests that the same development process, *engineer requirements*, is used at each level, although the explanation given above indicates that the work involved is different at each level. This apparently strange way of describing the process is used to introduce the fact that there is, in fact, a significant degree of commonality in the work done at each level. The purpose of this chapter is to explore these common aspects and to present a generic process that not only addresses the common aspects but also enables the different aspects to be accommodated.

It is important to stress that in a multi-level development, each level of development demands relevant expertise. At the higher levels, domain knowledge in the problem domain is vital. At the system level, it is important that a system-wide view is taken to avoid too narrow an interpretation of the stakeholder

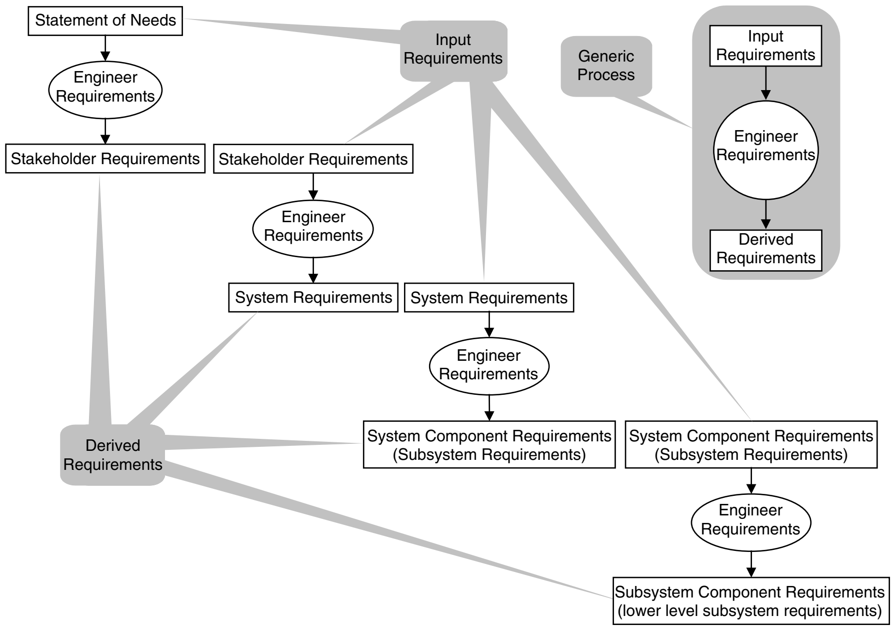


Figure 2.3 Identifying input and derived requirements of the generic process.

requirements. At this level there will inevitably be a solution bias introduced. People or organizations with a proven track record in the development of similar systems are necessary. Similarly, the subsystem developers will bring their own domain experience for the particular specialist area of their subsystem.

Hence it is unlikely that the same people will undertake development at every level. Even when the same organization is working on several levels, it is likely that different people will be involved, often from different departments. Therefore, it is useful to introduce the idea that each level of development is done in response to a "customer" at the level above, and will involve "suppliers" at the level below.

2.3.1 Input Requirements and Derived Requirements

Figure 2.3 shows an alternative view of Figure 2.2 in which the individual processes have been separated. This emphasizes that the requirements derived by one process become the input requirements of another process and leads naturally to the idea that the generic *engineer requirements* process takes in *input requirements* and generates *derived requirements* (also as shown in Figure 2.3).

2.3.2 Acceptance Criteria and Qualification Strategy

Before moving on to explain the internal details of the *engineer requirements* process, it is necessary to consider another class of information that is both an input to the process and derived by the process. This is information concerning the qualification strategy for the requirements.

To understand fully the significance of requirements and come to a satisfactory agreement that the requirements form a good basis for development, it is necessary to consider how the requirements will be demonstrated when the system (or component) has been implemented. This is partly achieved by determining, for each requirement, the criteria that will be used to establish whether or not the system that claims to implement the requirement is acceptable to the customer.

It is also necessary to determine the circumstances under which the criteria will be examined. In Chapter 1 the notion of test plans at each level was introduced. Testing is just one type of qualification strategy. Others include trials, certification and inspections. The type of qualification strategy to be used will depend on the nature of the system; for example, systems that have safety critical aspects will have to be checked much more carefully than, say, a management information system.

The full context of the *engineer requirements* generic process is therefore as shown in Figure 2.4.

The qualification strategy often introduces new requirements for test equipment, the use of existing facilities (*e.g.* wind tunnels, anechoic chambers) and special diagnostic functions or monitor points. In some circumstances a whole new project may evolve to develop the test equipment and other facilities required. For example, in avionics development it is necessary (for cost and safety reasons) to perform as much testing as possible before the equipment is installed in an aircraft. Even when it is installed, it will also be necessary to run with simulations prior to flight trials. Clearly, the test pilot must be assured that the avionics will perform to a known standard prior to first flight.

At lower levels in the hierarchy where items are to be manufactured, the qualification strategy may consider issues such as whether the supplier or the customer is responsible for the testing of each item supplied. Possible strategies include full testing of every item prior to delivery, batch testing by the supplier and possible random checks by the customer.

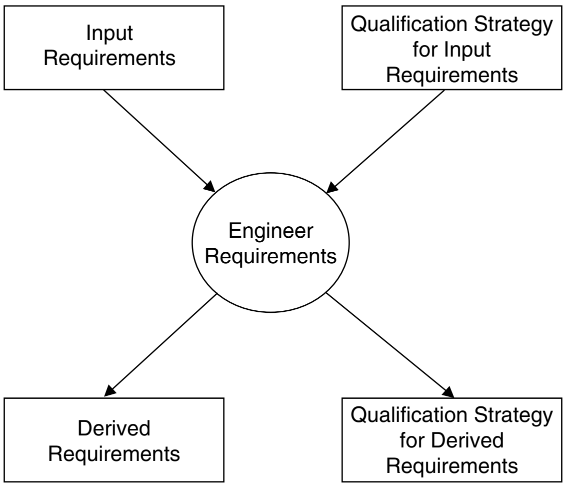

Input Requirements
Qualification Strategy for Input Requirements
Engineer Requirements
Derived Requirements
Qualification Strategy for Derived Requirements

Figure 2.4 Qualification strategy is essential.

2.4 Generic Process Introduction

Having established the context for the generic process, it is now possible to look inside the *engineer requirements* process. The process is introduced first in an ideal world in which nothing ever changes and then with modifications to accommodate changes.

2.4.1 Ideal Development

The *engineer requirements* process for the ideal world is shown in Figure 2.5. The process commences with the need to agree the input information for the project with the customer at the level above. The second activity in the process is to analyze the input information and consider how to develop the outputs required. This activity, which often goes on in parallel with agreeing the requirements, almost always involves the creation of one or more models and leads to analysis reports that together provide a basis for the derivation of requirements and qualification strategy for the lower level supplier(s). These requirements must, when they are sufficiently mature, be agreed with the suppliers to form the basis for a contract for the lower level development.

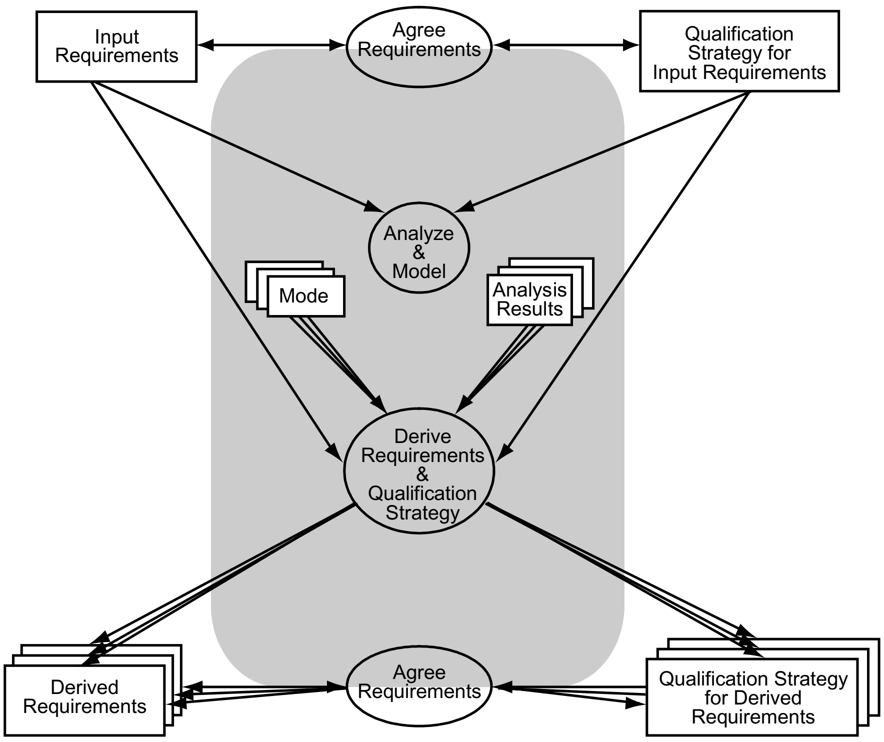

Input Requirements
Agree Requirements
Qualification Strategy for Input Requirements
Analyze & Model
Mode
Analysis Results
Derive Requirements & Qualification Strategy
Derived Requirements
Agree Requirements
Qualification Strategy for Derived Requirements

Figure 2.5 *Engineer requirements* process for an ideal world.

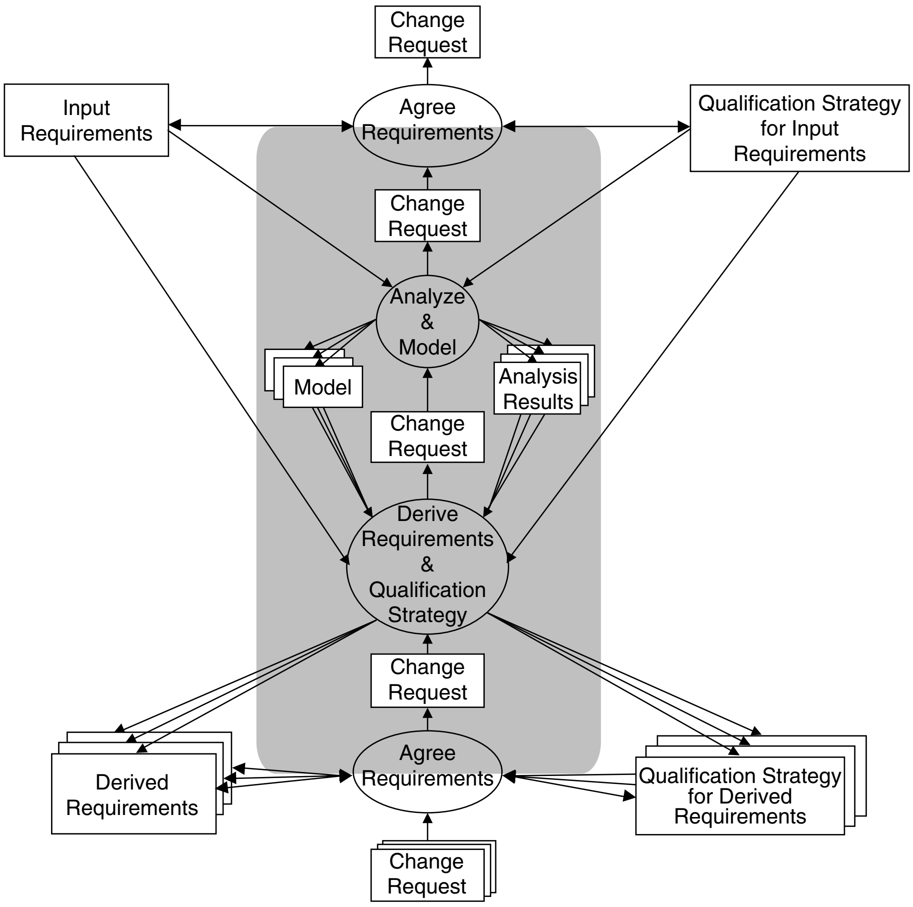


Figure 2.6 *Engineer requirements* process in context of changes.

Figure 2.5 also indicates that several sets of derived requirements may be generated. Each set must be agreed with the relevant supplier and some suppliers may be responsible for more than one component.

2.4.2 Development in the Context of Change

Unfortunately, the world hardly ever stands still. This is especially true in the arena of system development. It seems that everybody is constantly changing his or her mind or finding that what was previously agreed is no longer possible. Therefore, the generic process has to be modified, as indicated in Figure 2.6, to reflect this necessary evil.

The formality with which change is managed will depend on the nature and state of the project. During the early stages, changes can and must be made with ease so that progress can be made. However, there comes a time at which a commitment must be made and formal agreement struck. From this time, it is usual to have a more formal arrangement in which changes are not just inserted at the whim of anyone on the project. Instead, a process is used in which changes are first requested or proposed and then they are decided upon in the context of their impact on the project. The decision process will usually involve a person such as the project manager, who has the authority to make the decision supported as necessary by a group of people who constitute a change control board. Again, the degree of formality with which these people operate will depend on the nature of the project. The topic of change management is addressed in more depth in Chapter 8 in the context of project management.

In Figure 2.6, it can be seen that almost any activity can lead to the creation of a change and that these changes usually flow upwards. This does not mean that customers never change their minds or that the only problems discovered are lower level detail problems that flow from a top-down strategy. The situation is that the downward path is already accounted for in the normal flows, but the return path has to be explicitly catered for. One typical situation in which a change request might arise is, for example, that a limitation in a model or an anomaly in analysis results may well be discovered whilst attempting to generate a derived requirement or the qualification strategy for a derived requirement. A change request will recommend a modification to the model(s) and/or additional analysis work to investigate the problem. Similarly, a problem with in input requirement may be identified during the *analyze and model* process leading to the creation of a change request for the *agree requirements* process.

2.5 Generic Process Information Model

Before considering the subprocesses within the generic *engineer requirements* process, it is useful to introduce a generic information model that supports the process.

The diagrams used to represent the generic process contain both process symbols and data or information symbols. The diagrams indicate, via the arrows, which information is being generated and used by each process.

The purpose of an information model is to indicate what types of information exist and whether relationships can or should exist between the items of information. It is also useful to introduce state transition diagrams to indicate how the state of each type of information can be changed as time proceeds. Consequently, these state transition diagrams can give a visual indication of when and how processes interact with each other via the information.

2.5.1 Information Classes

Information types already encountered in the generic process context include:

- input requirement;
- derived requirement;

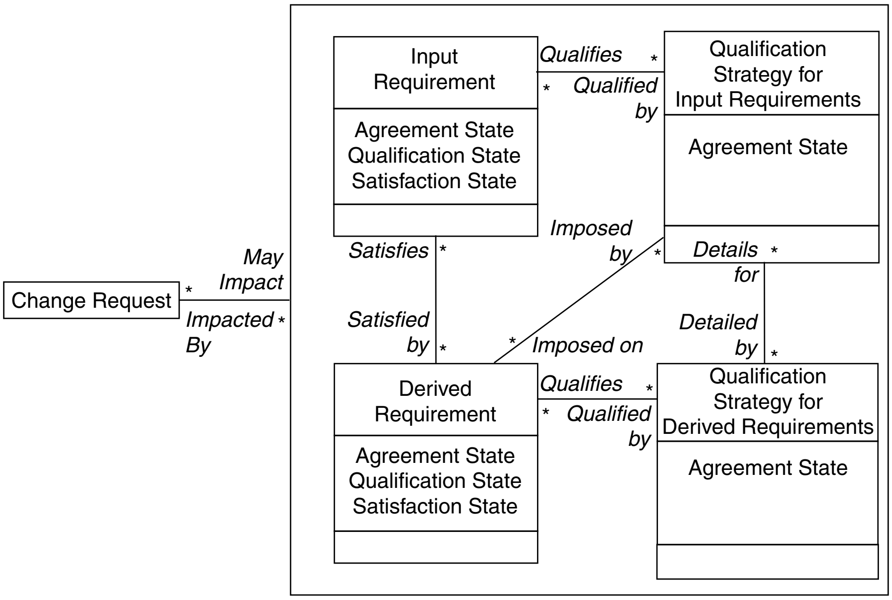


Figure 2.7 Information model for the generic process.

- qualification strategy for input requirements;
- qualification strategy for derived requirements;
- change request.

Figure 2.7 shows these five types of information expressed as a Unified Modelling Language (UML) class diagram. The name of the class is always shown in the uppermost section (or only section) of the class symbol. The middle section (if present) indicates the names of attributes that the class can have. The bottom section (if present) contains any operations (often called "methods") that can operate on the class.

The lines connecting the class symbols show relationships between classes, and these are called "Associations" in the UML. Thus an *input requirement* can be related to a *derived requirement* by a "satisfied by" relationship. Similarly, the *derived requirement* can be related to an *input requirement* by the inverse "satisfies" relationship. (These labels are known as "roles" in the UML.) The asterisk indicates that zero or more instances of the class can be involved in the association. Asterisks at both ends indicate that the association can be many to many. Thus, in the model in Figure 2.7 zero or more *input requirements* can be satisfied by a *derived requirement* and an *input requirement* can be satisfied by zero or more *derived requirements*. Some readers may question the zero lower limit, because it suggests that it is not necessary to have any association. However, if the lower limit were set to one, this would mean that an *input requirement* could not exist unless it was associated with at least one *derived requirement*. Clearly this is an impossible situation. It is essential that *input requirements* can exist prior to *derived requirements* being generated. Consequently, this is a reasonable model, because there may be times during a project when there will be no links between *input requirements* and *derived requirements* – for example, early in the development before the links have been established. However, a project manager would expect that there were links established as soon as possible. This would then indicate that progress had been made and that all *derived requirements* were justified by being there to satisfy an *input requirement* and, conversely, that all *input requirements* had been satisfied.

The qualification strategy classes can each qualify the appropriate type of requirement and the qualification strategy for the *derived requirements* can provide more details of an *input requirement* qualification. This can occur, for example, in safety critical systems where it may be necessary to perform lower level detailed inspections that contribute to the satisfaction of the higher level qualification criteria.

As mentioned earlier, it is possible that a qualification strategy may lead to the creation of special test rigs. This would be an example of the *imposed on* relationship between the qualification strategy for an *input requirement* and one or more *derived requirements*. Further examples of this relationship occur when, in order to be able to check a component, it is necessary to provide a monitor point. Such monitor points are often essential to be able to check the performance (speed, response, throughput, etc.) of a system under operational conditions.

A *change request* can apply to any of the other four classes. Enclosing the four classes inside an outer rectangle and making the relationship line touch this outer rectangle indicates this.

The middle section of the class symbols is used to define attributes that the class will have. The requirement classes each have the three attributes:

- agreement state;
- qualification state;
- satisfaction state.

These are defined in the following sections by means of statechart diagrams. The agreement state of the qualification classes is assumed to have the values *agreed* or *not agreed*.

2.5.2 Agreement State

The state chart for the agreement state is shown in Figure 2.8. In this type of diagram each (rounded) rectangle represents the state of a single requirement at some point in its history. The rectangle labelled *Being assessed* is known as a "super-state" because it contains other states within it. The lines connecting one state to another indicate transitions that cause the state to change.

The requirement state starts off in the *Proposed* state. When the customer is content that the requirement is sufficiently well formulated to be sent to the supplier, it is sent. The agreement state then enters the *Being assessed* super-state. During this state, the customer and supplier negotiate until an agreed requirement emerges.

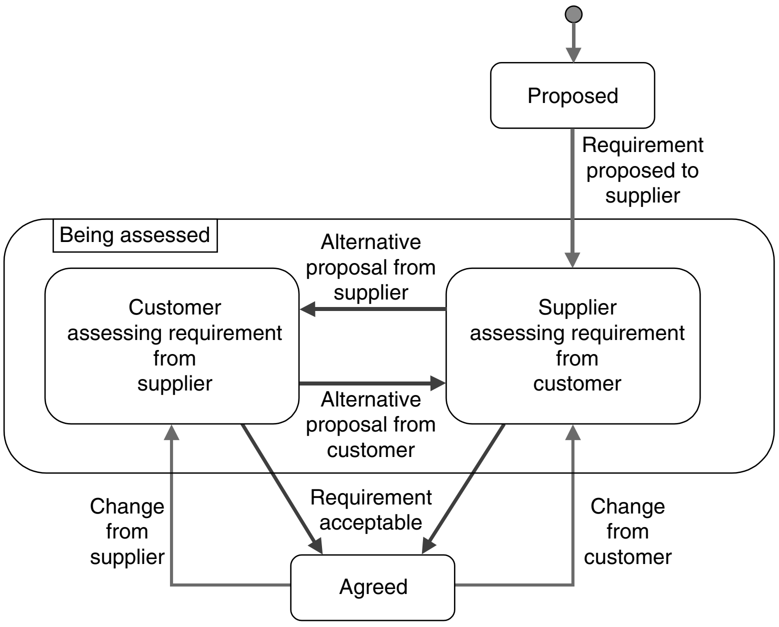

Proposed
Requirement proposed to supplier
Being assessed
Alternative proposal from supplier
Customer assessing requirement from supplier
Supplier assessing requirement from customer
Alternative proposal from customer
Change from supplier
Requirement acceptable
Change from customer
Agreed

Figure 2.8 Statechart for agreement state.

Once in the *Agreed* state, the requirement will stay there until either the customer or the supplier creates a change request. When this happens, the requirement's state re-enters the *Being assessed* state until a new agreed requirement emerges.

Within the *Being assessed* state, the customer and supplier take turns to suggest alternative forms of the requirement until an agreement is reached. The agreement state will therefore be in one of the two states shown depending on which party is currently making the assessment.

2.5.3 Qualification State

The qualification state of a requirement is shown in the statechart in Figure 2.9. The initial state is that there is *No qualification strategy decided*. When the qualification strategy has been agreed, the state can proceed to the state *Qualification strategy decided*. This state can then remain until a change request is received. The change may be directed either at the requirement itself or at the qualification strategy associated with it. When a change is requested, the state becomes *Qualification strategy suspect* until the impact of the change has been assessed. This assessment determines whether the existing qualification strategy can stand, and the state can return to *Qualification strategy decided*, or whether an alternative strategy must be decided, in which case the state becomes *No qualification strategy decided*.

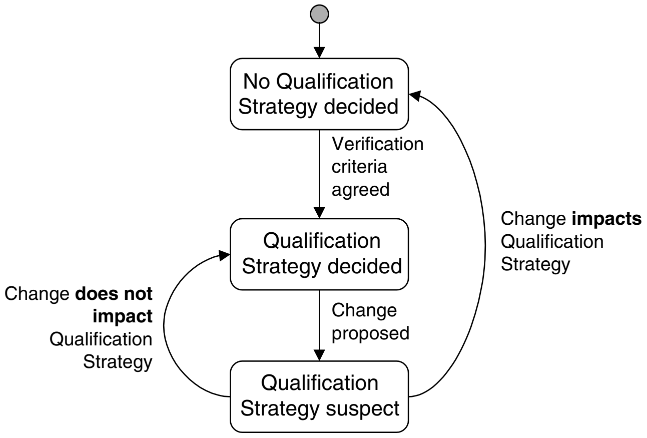

No Qualification Strategy decided
Verification criteria agreed
Qualification Strategy decided
Change proposed
Qualification Strategy suspect
Change **does not impact** Qualification Strategy
Change **impacts** Qualification Strategy

Figure 2.9 Qualification state.

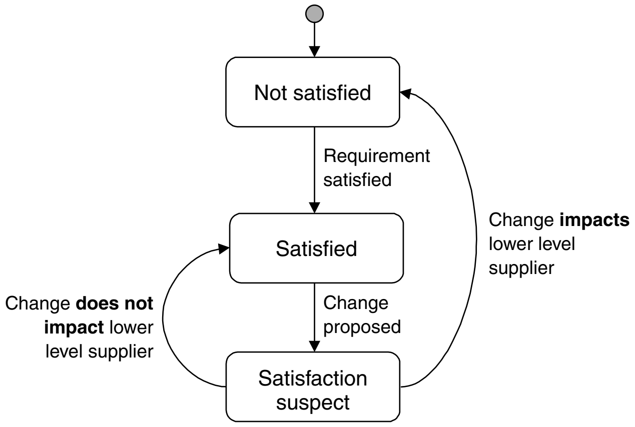

Not satisfied
Requirement satisfied
Satisfied
Change proposed
Satisfaction suspect
Change **does not impact** lower level supplier
Change **impacts** lower level supplier

Figure 2.10 Satisfaction states.

2.5.4 Satisfaction State

The statechart for the satisfaction state is shown in Figure 2.10. The logic of this state is very similar to the qualification states. The starting point is the *Not satisfied* state indicating that no *derived requirements* have been related to this requirement. When the *input requirement* has been satisfied by one or more *derived requirements*, the lower level supplier agrees the requirement and the higher level (customer) agrees that the *derived requirements* will, indeed, satisfy the *input requirement*, the state can be moved to the *Satisfied* state. It should be noted that there might be many *derived requirements* that have to be agreed before each single *input requirement* can achieve the *Satisfied* state.

When a change is proposed, the satisfaction state immediately becomes “Satisfaction suspect” irrespective of whether the proposed change is directed at the higher or lower level requirements. This suspect state is retained until the impact of the proposed change has been assessed and the satisfaction state can then become *Not satisfied* or *Satisfied*.

2.5.5 Information Model Constraints

Change requests bind together the agreement, qualification and satisfaction states. Registering a change request immediately changes all three states and requires additional work, first to determine whether there is any impact and second to address the consequences, if any, of the impact. Note that the satisfaction state can ripple up and down the requirements that are the subject of the satisfaction relationship. This ripple effect establishes the potential extent of any consequential change, that is, the “impact” of the change.

The agreement state of *derived requirements* must be consistent with the satisfaction state of *input requirements*, since an *input requirement* cannot achieve its *satisfied* state until the lower level supplier has agreed all of the *derived requirements* that satisfy it.

2.6 Generic Process Details

2.6.1 Agreement Process

The agreement process is always a concurrent activity between a supplier at one level and the customer at the level above as indicated in Figure 2.11.

Before any derivation work can commence, it is necessary to assess the *input requirements* to ascertain whether they form an adequate basis for the development to proceed.

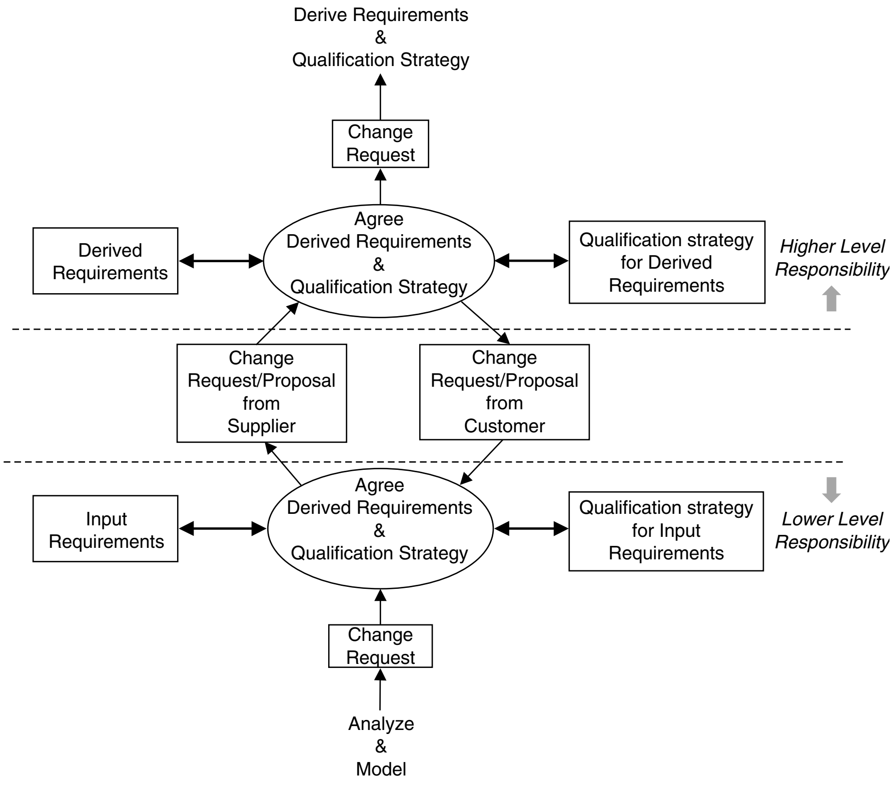
The assessment must answer the questions:

- Is the requirement complete?
- Is the requirement clear?
- Is the requirement implementable?
- Is the qualification plan clear and acceptable?

Potential answers to these questions lead naturally to the following reasons why a requirement may be rejected:

- Missing information – *e.g.* placeholders such as “TBA” (to be agreed), “TBC” (to be completed) or “TBD” (to be decided) may be used.
- Lack of clarity – ambiguity, contradiction, confusion, etc.
- Impossible to implement – no known solution.
- Unacceptable qualification plan.

Following the review, if a requirement and its qualification plan are acceptable, the status can be set to “Agreed”.



Derive Requirements & Qualification Strategy
Change Request
Derived Requirements
Agree Derived Requirements & Qualification Strategy
Qualification strategy for Derived Requirements
Higher Level Responsibility
Change Request/Proposal from Supplier
Change Request/Proposal from Customer
Input Requirements
Agree Derived Requirements & Qualification Strategy
Qualification strategy for Input Requirements
Lower Level Responsibility
Change Request
Analyze & Model

Figure 2.11 The agreement process.

If the requirement is not acceptable, then an alternative form is sent to the customer and the onus passes to the customer, and the agreement state (see Figure 2.8) becomes "Customer assessing requirement from supplier". If the customer is content with the alternative wording, then he can set the state to "Agreed". If not, then he or she proposes a further alternative and sends it to the supplier. The agreement state becomes "Supplier assessing requirement from supplier", and the onus returns to the supplier.

This process of proposal and counter proposal continues until an agreement is reached. Of course, it is possible that agreement may never be reached and a dispute emerges.

When either party proposes a change the "Being assessed" super-state is entered with the onus on the party receiving the change. Negotiation follows as described earlier until a new agreed form can be reached.

During the agreement process, change requests may be generated by the customer side to request that the *derived requirement* is modified. These will pass to the *derive requirements* and *qualification strategy* process so that the effect of the change can be assessed and, where necessary, adjustments made to one or more of the *derived requirements*. Of course, it can happen that the change cannot be handled completely at this level and the change may have to be escalated to the analyze and model process. This need to escalate the decision process up through the levels makes it imperative that people are working at each level. In other words, it is necessary to work concurrently on several levels simultaneously. This need completely destroys the notion of the “waterfall” lifecycle in which a sequence of activities takes place in a strict top-down order. Instead of a sequence of activities, development takes place as a concurrent set of negotiations and decision taking.

In many projects, the acceptance criteria and qualification plans are only decided fairly late. This can be well after the requirements themselves have been agreed and, in some cases, agreement is only reached just prior to the commencement of testing. This is very bad practice and usually leads to delays caused by late changes in requirements to make them testable!

2.6.2 Analyze and Model

This analysis part of this process is primarily concerned with understanding the nature and scope of the input requirements to assess the likely risks involved in satisfying them. Analysis work can range from feasibility studies to explore potential implementation options to the building of prototypes of some vital or high-risk components. It is often necessary to build performance models to investigate potential throughput and response figures.

The other uses of models in this process are to understand the nature of and provide a structure for the derived requirements. The most common models for understanding and structuring stakeholder requirements are use cases or user scenarios. These help to understand how people will use the intended system.

The most common models for structuring solutions in the solution domain are design architectures. These identify elements of the solution and indicate how they interact.

In many cases, the model is used to establish the design architecture of the proposed solution. These models are frequently obvious for well-established development domains (*e.g.* automobiles, telecommunications, aircraft) where a de facto architecture exists. However, for innovative developments where there is no established architecture, the model may be more abstract to allow for potential alternatives.

In general, the models used will depend entirely on the nature of the development that is being undertaken. As indicated earlier, the types of models used are very much domain specific. In software systems it is increasingly the case that object models are used. Table 2.1 indicates different sorts of models used in three industrial domains.

The point of developing the models is to understand the *input requirements* together with the proposed qualification strategy and experiment with alternative solution options prior to deciding how to proceed with the creation of *derived requirements*. This work will also consider possible qualification strategies for the *derived requirements* and this, in turn, may lead to the creation of requirements for test equipment and/or software. It can also lead to the identification of qualification requirements for the *derived requirements*.

Table 2.1 Examples of modelling techniques

Aircraft Industry
Aerodynamic model
Three-dimensional spatial model
Weight distribution model
Flight simulator
Rail industry
Timetable simulation
Safety, reliability and maintainability models
Car industry
Styling model
Dashboard model
Aerodynamic model

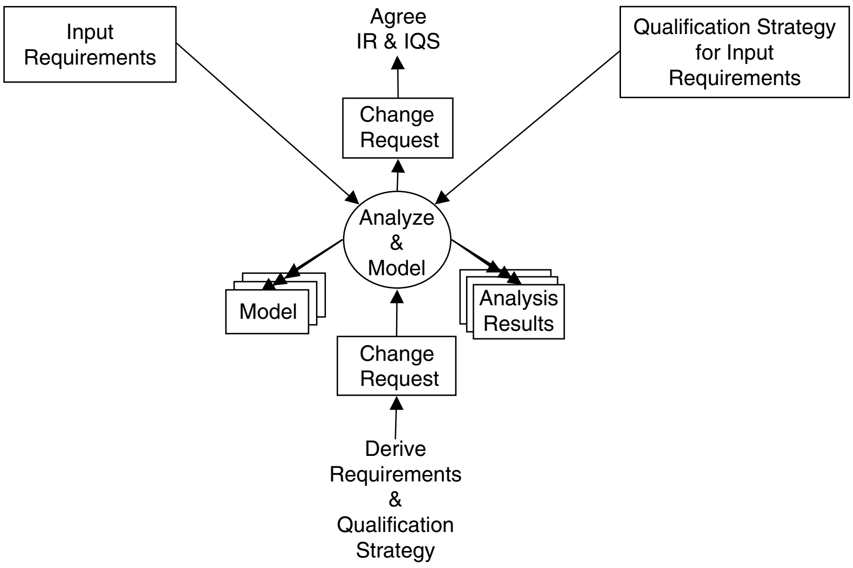


Figure 2.12 *Analyze and model* process.

The *analyze and model* process (Figure 2.12) can be undertaken in parallel with the agree process since it is likely to generate deeper insight into the nature of the requirements.

In Chapter 3, some widely used modelling techniques are reviewed, especially considering those used in the software industry. Chapter 5 explains how to use user scenario models to aid the understanding of stakeholder requirements and Chapter 6 considers function-oriented models that help to provide a framework for system requirements.

During the *analyze and model* process, it is likely that further questions will arise concerning the meaning and formulation of *input requirements*. This gives rise to change requests, which cause the *agree requirements* process to be re-entered.

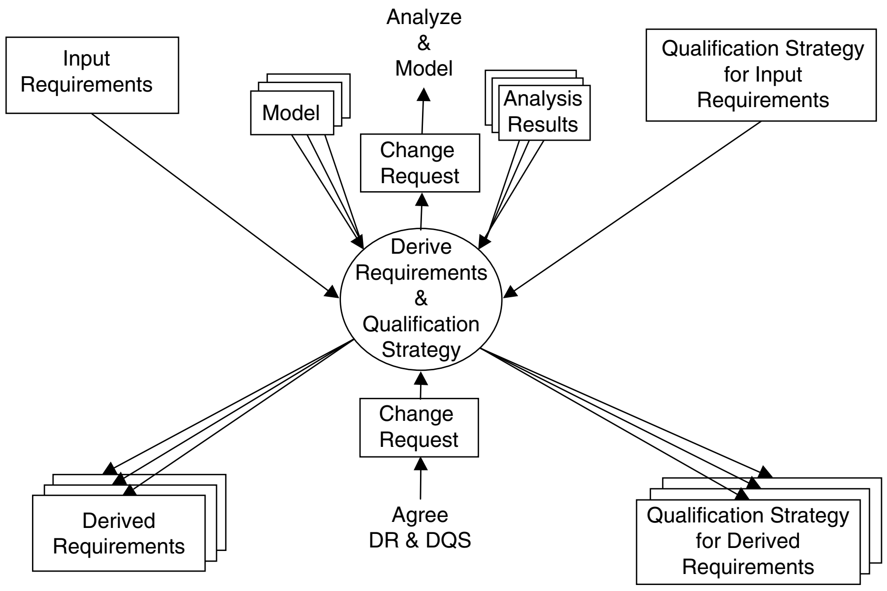

Input Requirements
Model
Analyze & Model
Change Request
Analysis Results
Qualification Strategy for Input Requirements
Derive Requirements & Qualification Strategy
Change Request
Agree DR & DQS
Derived Requirements
Qualification Strategy for Derived Requirements

Figure 2.13 Derive requirements and qualification strategy process.

2.6.3 Derive Requirements and Qualification Strategy

This process is illustrated in Figure 2.13.

Deriving Requirements

The way in which the models are used for this purpose varies, but the simplest one to consider initially is the derivation of component requirements based on a design architecture. Here it is possible to determine the specific requirements that must be satisfied by each component. Some of these requirements may be identical with one or more *input requirements*; others may have been derived from *input requirements* in order to partition them amongst the components. A further set of requirements consists of constraints imposed either by the component architecture or *input requirements*. These constraints include interface constraints and possible physical constraints such as mass, volume, power usage and heat dissipation.

In practice, some work on the allocation or derivation of requirements for components may proceed in advance of final agreements on the *input requirements* and their qualification strategy. However, it is not possible to *complete* this activity prior to final agreement.

In addition to establishing the component requirements, it is also necessary to establish the satisfaction relationship between the *input requirements* and the *derived requirements*. This relationship indicates which *input requirements* are satisfied by which *derived requirements* and can be used to establish that:

- all *input requirements* are satisfied;
- all *derived requirements* are necessary (i.e. they directly or indirectly satisfy one or more *input requirements*).

It is not sufficient just to assert that a satisfaction link exists, as for example in a cross-reference matrix. The justification for each link should also be stated. These justification statements constitute a satisfaction argument.

During the process of generating requirements from the models, it may become clear that there is a defect or an omission in one or more of the models. This causes a change request to be issued back to the modelling team, who will then either modify the model directly or ask for further clarification or change to *input requirements*. Thus the change escalation process continues.

Deriving the Qualification Strategy

As discussed above, the satisfaction relationship is about generating *derived requirements* from *input requirements* – how the system is designed. In contrast, the qualification strategy plans how each requirement will be tested at each level.

The qualification strategy consists of a set of qualification actions, each one a particular kind of trial, test or inspection. There may be several qualification actions defined against each requirement.

Each qualification action should take into account the following aspects:

- the ***kind*** of action that would be appropriate for the requirement;
- the ***stage*** at which each action could take place – the earlier the better;
- any special ***equipment*** that would be needed for the action;
- what would constitute a successful ***outcome***.

The qualification plan may be structured according to either the stage or the type of action.

The qualification actions defined should be appropriate to the level of requirements. In other words, stakeholder requirements give rise to acceptance trials, whereas system requirements give rise to system tests, that is, prior to delivery to the customer. It is not necessary to define system tests against stakeholder requirements, since those system requirements derived from the stakeholder requirement will have their own system tests.

Take, for instance, the example shown in Figure 2.14, in which a system requirement for a ship is decomposed into two requirements on different subsystems, the hull and the propulsion system. Two qualification tests are planned against the system-level requirement and two more against the subsystem requirements.

Therefore, for a full understanding of how a requirement will be tested, both the satisfaction relationship and the qualification strategy are necessary. To understand the qualification status of a high-level requirement, the results of qualification actions against requirements that flow down from it at all levels have to be taken into account, by making use of both the satisfaction and the qualification relationship.

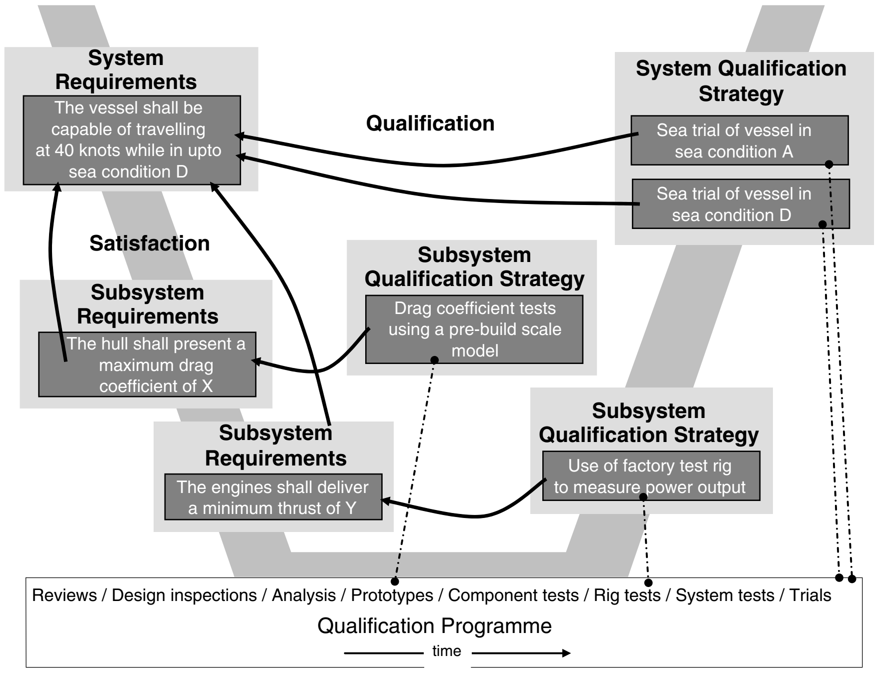

System Requirements
The vessel shall be capable of travelling at 40 knots while in upto sea condition D
Qualification
System Qualification Strategy
Sea trial of vessel in sea condition A
Sea trial of vessel in sea condition D
Satisfaction
Subsystem Qualification Strategy
Drag coefficient tests using a pre-build scale model
Subsystem Requirements
The hull shall present a maximum drag coefficient of X
Subsystem Requirements
The engines shall deliver a minimum thrust of Y
Subsystem Qualification Strategy
Use of factory test rig to measure power output
Reviews / Design inspections / Analysis / Prototypes / Component tests / Rig tests / System tests / Trials
Qualification Programme
time

Figure 2.14 Qualification information.

2.7 Summary

A generic process that can be simultaneously applied at each level in a system development has been presented. The benefit of this generic process is that it identifies common actions that are relevant at every level:

- agreeing *input requirements* with the customer;
- analysis of *input requirements* to determine the risks and potential pitfalls in satisfying the requirements;
- creating one or more models to investigate possible strategies for deriving requirements;
- generating requirements derived from the *input requirements* via *the analyze and model* information;
- agreeing the *derived requirements* with the team(s) that will be responsible for implementing them;
- establishing the satisfaction relationship between *input requirements* and *derived requirements*;
- establishing the qualification relationship between *derived requirements* and the relevant qualification strategy.

These actions lead to the establishment of information according to the information model presented. The current state of the information can be used to measure progress, to assess the impact of proposed changes and to define metrics on how a project is performing. For example, the state of a requirement can be captured by its three attributes:

- agreement;
- satisfaction;
- qualification.

The ideal state for any requirement in any system development is that it should be:

- agreed between customer and supplier;
- have a qualification strategy agreed for it;
- be satisfied by lower level requirements (or design).

The extent to which a project's requirements deviate from this ideal state represents the degree of risk to which the project is exposed from the requirements management point of view and also indicates the extent of the work necessary to get the requirements into the ideal state.