2 THE NFR FRAMEWORK IN ACTION

Consider the design of an information system, such as one for managing credit card accounts. The system should debit and credit accounts, check credit limits, charge interest, issue monthly statements, and so forth.

During the development process of requirements elaboration, systems design and implementation, a developer needs to make decisions such as:

- How frequently will account information be updated?
- How will customer identity be validated — e.g., by using personal identification numbers (PIN codes) or biometrics?
- Will a certain group of data be stored locally or replicated over multiple sites?

These development decisions have important implications for the security, performance, accuracy, cost and other aspects of the eventual system. The significance of these *non-functional requirements* (or *software quality attributes*) are widely recognized. Attaining software quality attributes can be as crucial to the success of the system as providing the functionality of the system. For example, inaccurate credit account information can lead to monetary loss and damage to the reputation of a financial institution, while poor response time could lead to poor morale and eventually loss of customers.

Most conventional approaches to system design are driven by functional requirements. Developers focus their efforts primarily on achieving the desired functionality of the system – calculating account interest, issuing monthly statements, etc. Although decisions about *how* to achieve functionality are made along the way, usually with non-functional requirements (such as cost and performance) in mind, these considerations may not be systematic, and may often not be documented. Furthermore, these software quality attributes may often be viewed as *consequences* of the decisions, but not something that the developer can strive for in a coherent, well thought-out way.

Note that in this section we are not using the terms "design" or "development" to refer to a particular phase of development. Rather, they are used in the broad sense of the process of developing a target artifact by starting with a source specification and producing *constraints* upon the target artifact. This could occur at various phases of development (e.g., requirements specification, conceptual design, or implementation).

2.1 USING THE NFR FRAMEWORK

In contrast to functionality-driven approaches, the NFR Framework uses non-functional requirements such as security, accuracy, performance and cost to drive the overall design process. The Framework aims to put non-functional requirements foremost in the developer's mind.

There are several major steps in the design process:

- acquiring or accessing knowledge about:
  - the particular domain and the system which is being developed,
  - functional requirements for the particular system, and
  - particular kinds of NFRs, and associated development techniques,
- identifying particular NFRs for the domain,
- decomposing NFRs,
- identifying "operationalizations" (possible design alternatives for meeting NFRs in the target system),
- dealing with:
  - ambiguities,
  - tradeoffs and priorities, and
  - interdependencies among among NFRs and operationalizations,
- selecting operationalizations,
- supporting decisions with design rationale, and
- evaluating the impact of decisions.

These are not necessarily sequential steps, and one may also need to iterate over them many times during the design process. A developer may choose refinements, having operationalizations in mind; thus the development process may move up and down, rather than being strictly top-down.

It would be extremely helpful, if at each step in the process, the developer could draw on available knowledge that is relevant to that step in the process. This is precisely what the NFR Framework aims to provide. The Framework offers a structure for representing and recording the design and reasoning process in graphs, called *softgoal interdependency graphs (SIGs).* The Framework also offers *cataloguing of knowledge* about NFRs and design knowledge, including development techniques. By providing SIGs and drawing on catalogues, the contextual information at each step can be used to trigger and bring forth previously-stored knowledge to help the developer carry out that step.

Softgoal Interdependency Graphs

The operation of the Framework can be visualized in terms of the incremental and interactive construction, elaboration, analysis, and revision of a *softgoal interdependency graph* (*SIG*). The graph records the developer's consideration of *softgoals,* and shows the *interdependencies* among softgoals.

Major concepts of the Framework appear in the graphical form in SIGs. Softgoals, which are "soft" in nature, are shown as clouds. Main requirements are shown as softgoals at the top of a graph. Softgoals are connected by *interdependency links,* which are shown as lines, often with arrowheads. Softgoals have associated *labels* (values representing the degree to which a softgoal is achieved) which are used to support the reasoning process during design. Interdependencies show *refinements* of "parent" softgoals *downwards* into other, more specific, "offspring" softgoals. They also show the *contribution* (impact) of offspring softgoals *upwards* upon the meeting of other (parent) softgoals.

To determine whether softgoals are achieved, an *evaluation procedure (labelling algorithm)* is used, which considers labels and contributions, and, importantly, decisions by the developer.

It is important to note that the *developer* has control over what softgoals are stated, how they are refined, and the extent to which they are refined. The design process and evaluation procedure are *interactive.* Evaluation is also "semi-automatic," i.e., assisted by a procedure (algorithm), but with the developer in control.

Cataloguing Design Knowledge

A very important aspect of the Framework is that developers are able to draw on an organized body of design knowledge (including development techniques) that has been accumulated from previous experience. This type of knowledge can be arranged in knowledge *catalogues.*

There are three kinds of catalogues used. One kind of catalogue represents knowledge about the particular types NFRs being considered, such as security and performance, and their associated concepts and terminology. Another kind of catalogue is used to systematically organize *development tech-*

niques (methods), which are intended to help meet requirements, and are available to developers. The third type of catalogue shows implicit interdependencies *(correlations, tradeoffs)* among softgoals.

The design knowledge in catalogues may come from many sources. General knowledge for various areas (e.g., performance, security, usability) are typically available in textbooks, developer guides and handbooks. More specialized knowledge may be accumulated by specialists in industry and academia, or within an organization. Individual developers and teams also build up knowledge from their experiences over a number of projects that can be reused. By making this knowledge available in a single design framework, developers can draw on a broad range of expertise, including those beyond their own immediate areas of speciality, and can adapt them to meet the needs of their particular situations.

These catalogues, such as those offered in Part II, provide a valuable resource for use and re-use during development of a variety of systems. This was our experience in the case studies of Part III, where catalogued knowledge of domain information and NFRs was used throughout the design process. Thus other steps in the process can be aided by gathering and cataloguing knowledge *early* in the process.

Now we consider the various steps of the process of using the Framework.

2.2 ACQUIRING DOMAIN KNOWLEDGE

During the process, the developer will acquire and use information about the domain and the system being developed. This includes items such as *functional requirements,* expected organizational *workload,* and organizational *priorities.*

For a credit card system, for example, the functional requirements include operations to authorize purchases, update accounts and produce statements. Organizational workload includes the number of cardholders and merchants, and the expected daily volume of purchases. The credit card organization will have some priorities, such as emphasizing the fast cancellation of stolen cards, and the provision of fast authorization.

Of course the developer will need to acquire a *source specification* of the system, before moving towards a *target.* For example, the source might be a set of requirements, and the target might be a conceptual design. As another example, the developer might start with a conceptual design and move towards an implementation of the system.

The case studies of Part III include descriptions of the domains studied.

2.3 ACQUIRING AND CATALOGUING NFR KNOWLEDGE

The developer will be drawing on catalogues of knowledge of NFRs and associated development techniques.

To provide a terminology and classification of NFR concepts, *NFR type catalogues* are used. Figure 2.1 shows a catalogue of NFRs. The NFR types are arranged in a hierarchy. More general NFRs are shown above more specific ones. For example, performance has *sub-types* time and space, which in turn have their own sub-types. The NFRs shown in bold face are considered in detail in this book.

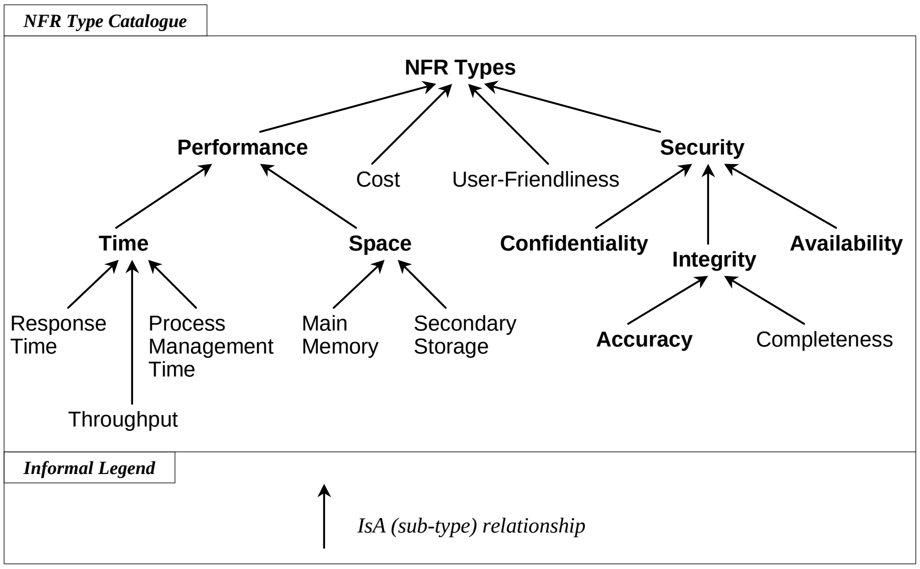


Figure 2.1. A catalogue of some NFR Types.

The NFR types provide a terminology to express requirements. For example, we can speak of the *security* of an account, or the *response time* for sales authorization.

Standard development techniques, along with methods of decomposing NFRs, are also organized into *method catalogues.* Interdependencies among NFRs (for example, stating that auditing helps security) are also organized into *correlation catalogues.* We will give examples of methods and correlations in this chapter, and will discuss these two types of catalogues in more detail in Section 2.12.

Normally, system developers can access existing catalogues from the start of development. The catalogues can be extended during development, to deal with additional or more refined concepts or development techniques which subsequently need to be addressed.

2.4 IDENTIFYING NFRs

In using the NFR Framework, one constructs an initial softgoal interdependency graph by identifying the main non-functional requirements that the particular system under development should meet. In the credit card system example, these may include security of account information, and good performance in the storing and updating of that information. These non-functional requirements (NFRs) are then treated as *softgoals* to be achieved, i.e., they are goals which need to be clarified, disambiguated, prioritized, elaborated upon, etc. This particular kind of softgoal is called an *NFR softgoal.* As we will soon see, NFR softgoals are one of three kinds of softgoals.

The developer will identify specific possible development techniques. From among them, the developer will choose solutions in the target system that meet the source requirements specification. Thus the developer begins by systematically *decomposing* the initial NFR softgoals into more specific *sub-softgoals* (or *subgoals*).

Let us consider the requirements to "maintain customer accounts with good security" and "maintain customer accounts with good performance."

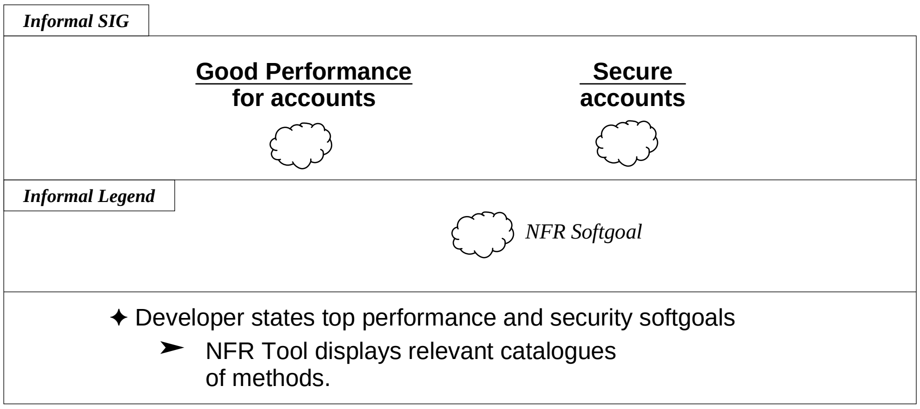


Figure 2.2. An initial Softgoal Interdependency Graph with NFR softgoals representing requirements for performance and security of customer accounts.

We are considering two non-functional requirements here, one for good performance for accounts, the other for good security of accounts. These non-functional requirements are represented as NFR softgoals Good Performance for accounts and Secure accounts. The NFR softgoals are represented by clouds, shown in the softgoal interdependency graph (SIG) of Figure 2.2.

Softgoals have an *NFR type,* which indicates the particular NFR, such as security or performance, addressed by the softgoal. In this chapter, the NFR types of some top-level softgoals are underlined in figures. Softgoals also have a subject matter or *topic,* here, accounts.

In this chapter, we use a syntax for SIGs which conveys the main points but is somewhat informal. Here the figure is an "informal" softgoal interdependency graph (informal SIG). A more precise syntax is introduced in Chapters 3 and 4, and is used in the remainder of the book.

Figures in this book have a *logo* in the top left corner (such as *Informal SIG*) indicating the kind of figure. A list of the different kinds of logos for figures is given in Figure 0.1.

Some figures also have legends to describe new symbols. A collected *Legend for Figures* appears at the front of this book.

Figures in this chapter also have an informal description of the process of developing SIGs. A diamond introduces an action by the developer. A right-arrow introduces a response to the developer, done by consulting catalogues and executing algorithms (procedures).

Responding to a developer's decisions, along with the drawing of SIGs, can be provided by an interactive design support tool, or can be done "by hand" by the developer using "pencil and paper." The responses in the figures of this chapter are suggestive of how a tool, such as the "NFR Tool" [Chung93a, 94c], could be used. In this book, however, we do not assume that a particular method is used to draw graphs or respond to developers' decisions. Rather, the presentation below will focus on the usage of the NFR Framework's concepts. This is done to offer some evidence that the Framework is useful. However, we do not make a claim about the ease of handling large SIGs for large systems

The following steps of using the NFR Framework can be viewed as an analysis of requirements, followed by a synthesis of operationalizations to meet the requirements. First, softgoals are broken down into smaller softgoals. We deal with ambiguities, and also consider domain information and priorities. Throughout the analysis we consider interdependencies among softgoals. Then we synthesize solutions to build quality into the system being developed. We consider possible alternatives for the target system, then choose some, and state reasons for decisions. Finally we see how well the main requirements have been met.

Interestingly, the NFR Framework is able to deal with different NFRs in one graph at the same time, even when the NFRs have different natures (here, performance and security). As we shall see, the NFR Framework can deal with interactions among these different NFRs.

2.5 DECOMPOSING NFR SOFTGOALS

In decomposing an NFR softgoal, the developer can choose to decompose its *NFR type* or its *topic*. In the example, the two softgoals share the same topic, accounts, but address different NFRs, performance and security.

Now we will decompose the two NFR softgoals of the example, starting with the security requirement.

The initial security requirement is quite broad and abstract. To effectively deal with such a broad requirement, the NFR softgoal may need to be broken down into smaller components, so that effective solutions can be found.

In addition, the requirement may be ambiguous. Different people may have different conceptions of what "security" constitutes in the context of credit card account information.

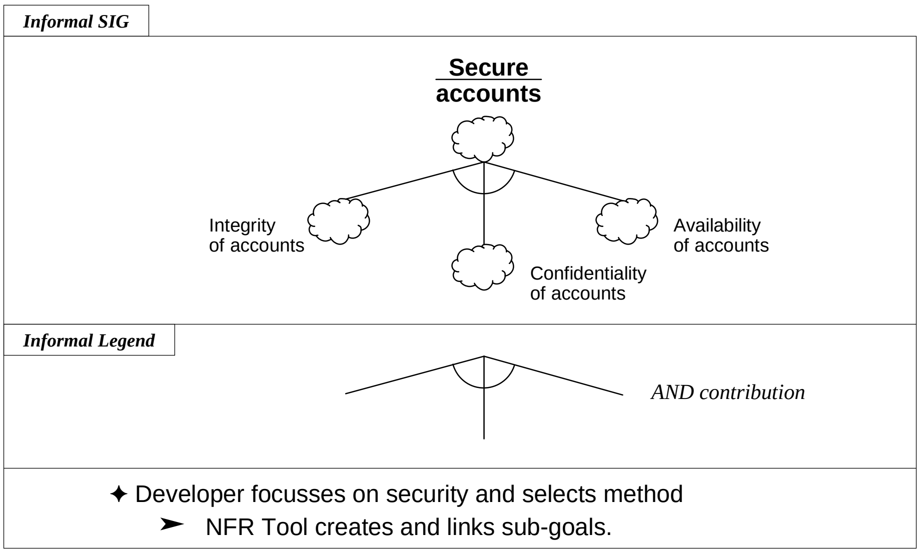


Figure 2.3. Decomposing NFR softgoals into more specific non-functional requirements.

By treating this high-level requirement as a *softgoal* to be achieved, we can decompose it into more specific subgoals which together "satisfice" (should meet) the higher-level softgoal. Thus the Secure accounts NFR softgoal can be decomposed into sub-softgoals for the

- integrity,
- confidentiality, and
- availability.

of the accounts. This is shown in Figure 2.3, which is an extension (downwards) of part of Figure 2.2. Such series of figures are used throughout this book to show the development of SIGs, where one figure builds on earlier ones.

In the graphical notation, clouds denote *softgoals* and lines represent *interdependencies* among softgoals.

Softgoals contribute, positively or negatively, to fulfilling other softgoals. There are different types of *contributions*. When *all* of several sub-softgoals are needed together to meet a higher softgoal, we say it is an *AND* type of contribution. Here, we say that *if* integrity, confidentiality *and* availability are all met, *then* as a group their contribution will be to achieve the security softgoal. This is an *AND* contribution. It is shown with lines grouped by an arc.

Typically, softgoals are shown as being *refined* downwards into sub-softgoals (subgoals), and subgoals *contribute* upwards to parent softgoals.

It is interesting to note that steps used in constructing SIGs can draw on catalogues, such as the NFR Type catalogue of Figure 2.1. In Figure 2.3, for example, an entry (Security) in the type catalogue has specialized types (Integrity, Confidentiality and Availability). And in the SIG we see the same pattern, where inter-connected (interdependent) softgoals have these same types.

If patterns are be expected to be re-used when building SIGs, *methods* can be defined and entered in a catalogue. Here, for example, a method can be defined which takes a parent softgoal of a particular type, and produces offspring softgoals which have sub-types of the parent's type. This SubType method is used in the SIG.

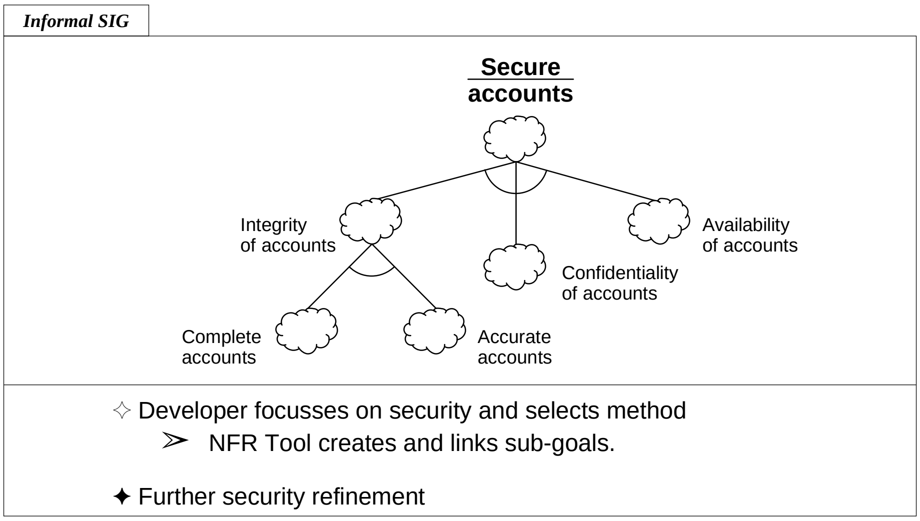
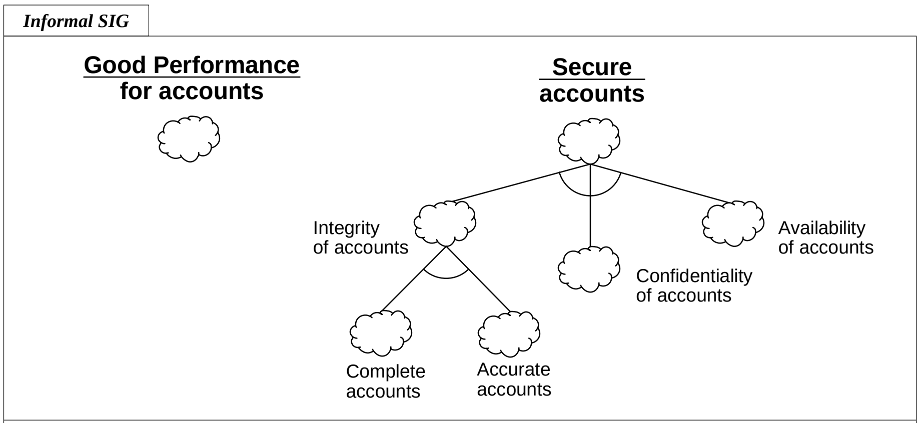


Figure 2.4. Further decomposition of a security softgoal.

The NFR softgoal that accounts will have integrity can be further decomposed into the subgoals that account information be complete, and accurate. This is shown in Figure 2.4. This is another application of the SubType method. Here, it considers the sub-types of Integrity from the catalogue of Figure 2.1.

In the descriptions at the bottom of figures in this chapter, solid diamonds and right-arrows represent new actions and responses, while outlined



✧ Developer states top performance and security softgoals
  ➢ NFR Tool displays relevant catalogues of methods.

✧ Developer focusses on security and selects method
  ➢ NFR Tool creates and links sub-goals.

✧ Further security refinement

✦ Developer focusses on performance softgoal for accounts
  ➤ NFR Tool displays catalogue of performance methods.

Figure 2.5. Considering a performance softgoal.

ones provide context by repeating actions and responses which were shown in earlier figures.

Recall that we started with two main NFRs, for security and performance of accounts. Figure 2.5 shows these initial NFRs from Figure 2.2, along with the security development of Figure 2.4.

Having refined the security requirement, the developer now focusses on the performance requirement.

The developer decides to decompose the performance softgoal with respect to its NFR type. This results, in Figure 2.6 (an extension of Figure 2.5), in two subgoals:

- one for good space performance for accounts, and
- one for good time performance for accounts.

Good time performance here means fast response time, and good space performance means using little space. Here the subgoals make an *AND* contribution to the performance softgoal.

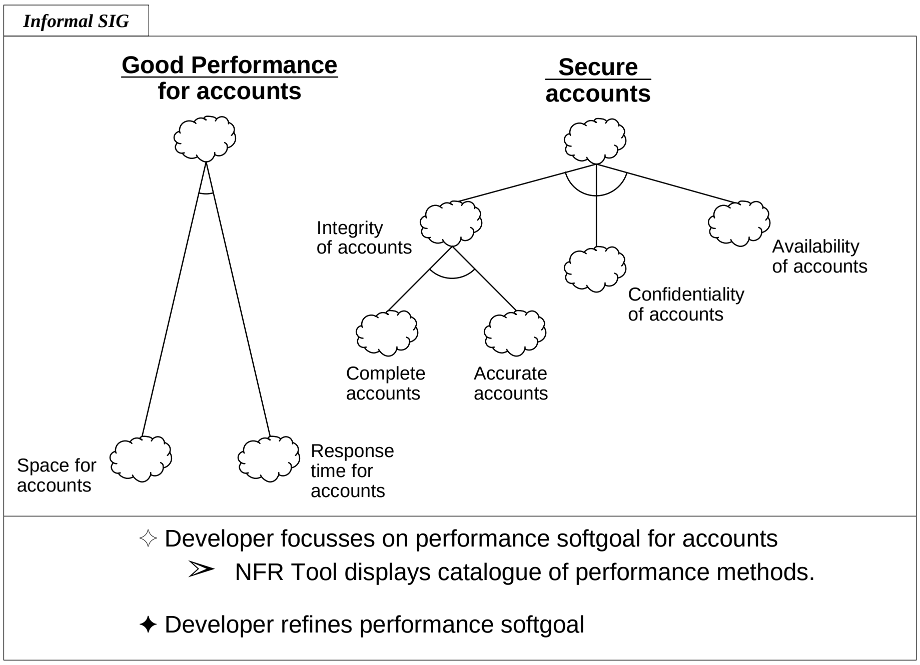


Figure 2.6. Decomposing a performance softgoal.

This is another use of the SubType method. It draws on the subtypes of Performance in the NFR type catalogue of Figure 2.1.

As well as their *NFR type*, softgoals can also be decomposed by their *topic*. For example, a performance softgoal for credit card accounts could be decomposed into performance softgoals for gold accounts and for regular accounts.

2.6 DEALING WITH PRIORITIES

Softgoal interdependency graphs can grow to be quite large and complex. How can a developer focus on what is important?

One way is to identify *priorities*. Extra attention can then be put towards meeting the priority softgoals.

Priorities can arise from consideration of several factors. These include *domain information* such as *organizational priorities* (e.g., a bank may consider security very important) and *organizational workload* (e.g., a credit card system may have a large number of sales to authorize each day). In addition, requirements can be identified as priorities during various phases of development.

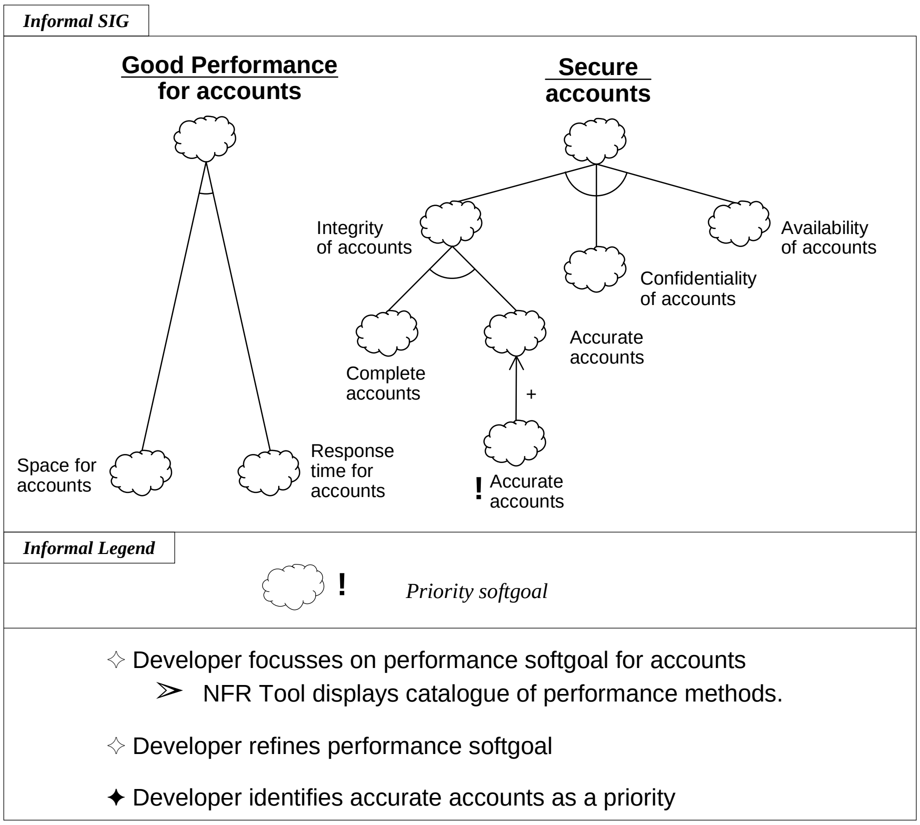


Figure 2.7. Identifying a softgoal as a priority.

Some priority softgoals will be identified as *critical*, as they are vital to the success of the system or organization. Other softgoals will be noted as being *dominant*, as they deal with a significant portion of the organization's workload. Priority softgoals are identified by an exclamation mark (!).

Figure 2.7 identifies Accurate accounts as being a priority softgoal. This is shown by producing an offspring with the same type and topic, but noted as a priority by "!".

The priority offspring contributes positively to the parent, and this is indicated by "+". This positive contribution is an example of a *contribution type,* which shows the impact of offspring softgoals upon their parent softgoal.

The reasons for prioritizing a softgoal can be noted as design rationale, discussed in Section 2.9 below. Now that the priority is identified, it can be analyzed and dealt with.

For example, priorities may be used to make appropriate *tradeoffs* among softgoals. As an example, fast authorization of credit card sales may be given higher priority than fast determination of travel bonus points to be given to gold cardholders. Here, the developer may perform the authorization before the bonus calculation.

Knowledge of tradeoffs can be captured in catalogues, and made available for re-use in dealing with softgoal synergy and conflicts. Thus throughout the development process, various tradeoff considerations can be made.

2.7 IDENTIFYING POSSIBLE OPERATIONALIZATIONS

While the refinement process so far provides more specific interpretations of what the initial NFRs of "secure" and "good performance" mean, it does not yet provide means for actually accomplishing security and performance for accounts.

At some point, when the non-functional requirements have been sufficiently refined, one will be able to identify possible development techniques for achieving these NFRs (which are treated as NFR softgoals) and then choose specific solutions for the target system. The development techniques can be viewed as methods for arriving at the "target" or "destination" of the design process.

However, it is important to note that there is a "gap" between NFR softgoals and development techniques. To get to the destination, one must bridge the gap. This involves performing analysis, and dealing with a number of factors. These include ambiguities, priorities, tradeoffs, and other domain information such as the workload of the organization. These factors will have to be addressed at various steps in the process.

We show how development techniques are identified here, and how they are selected in Section 2.10.

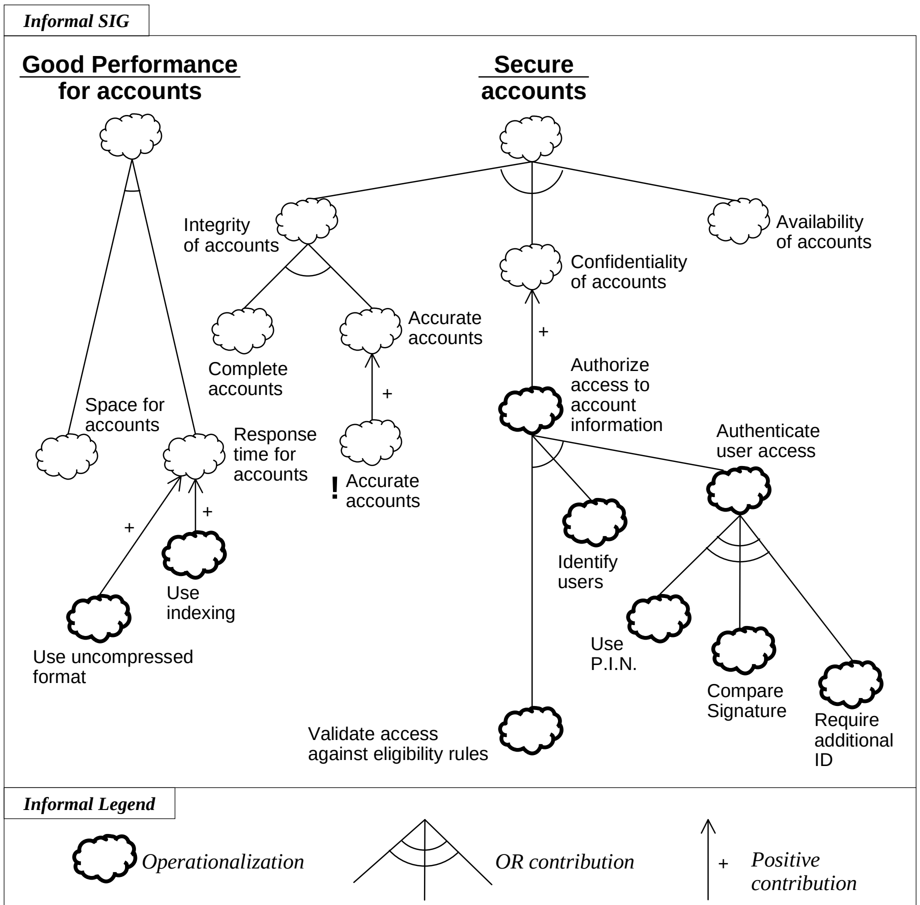
Let us consider the challenge of providing good Response time for accounts. One possible alternative is to use indexing (Figure 2.8). In this case, Use indexing is a development technique that can be implemented. It is a candidate for the task of meeting the response-time NFR. But it is no longer a non-functional requirement. This is contrasted with Response time, which is still a *software quality attribute*, i.e., a non-functional requirement, since it is not something that can be implemented directly.

We call the development techniques *operationalizations* of the NFR softgoals.

We say that indexing *operationalizes* response time. We also say that the response time NFR is *operationalized by* indexing.

Operationalizing softgoals are drawn as thick (dark) clouds, and are another kind of softgoal.

Note that operationalizations are not limited to operations and functions. Instead, operationalizations can correspond to data, operations and constraints in the target system.



✧ Developer identifies accurate accounts as a priority

✦ Developer focusses on response time, confidentiality and security
  ➤ NFR Assistant displays catalogue of possible techniques and trade-offs.

✦ Developer selects possible techniques
  ➤ NFR Assistant creates and links techniques.

Figure 2.8. Identifying possible operationalizations for NFR softgoals.

Like other softgoals, operationalizing softgoals make a *contribution,* positive or negative, towards parent softgoals. Here, the use of indexing helps meet the NFR softgoal for good response time. This positive contribution is shown by "+" in Figure 2.8. There are several contribution types. Earlier, we saw the *AND* contribution type.

In addition, the use of an uncompressed format makes a positive contribution towards meeting the response time requirement. However, as we will see later, it has a negative impact on space performance.

Note that we have not yet chosen which operationalizations will be used in the target system. What we can say now is that the use of either indexing or an uncompressed format, or both, will make a positive contribution towards achieving good response time.

Let us now consider how to meet the security requirement, particularly Confidentiality of accounts. One development technique is to allow only authorized access to account information. Authorize access to account information is an operationalizing softgoal which makes a positive contribution to confidentiality.

The operationalizations can be drawn from catalogues of development techniques, based on expertise in security, performance, and information system development. Catalogues can aid the search for possible operationalizations.

The transition from NFR softgoals to operationalizing softgoals is a crucial step in the process, because NFRs need to be converted into something that can be implemented. However, one may not be able to convert initial requirements into a concrete operationalization in one step, i.e., the initial operationalization (development technique) may not be specific enough. Often, there needs to be further refinements and elaborations. Furthermore, there can be different ways for refining or elaborating these general operationalizing softgoals, i.e., one needs to continue to identify other possible development techniques and choose among them. In the NFR Framework, we continue to treat these operationalizing softgoals – both general ones as well as increasingly specialized and specific ones – as softgoals to be addressed. This allows us to use the same systematic framework to decompose operationalizing softgoals into more specific ones.

For example, the operationalizing softgoal Authorize access to account information can be detailed (further operationalized) in terms of a combination of:

- identifying users,
- authenticating user access, *and*
- validating access against eligibility rules.

The *AND* contribution joining operationalizing softgoals means that all three offspring have to be achieved, in order to achieve the parent softgoal, Authorize access to account information. If any one is not achieved, the parent softgoal will not be achieved.

In turn, authenticating user access can be decomposed into any one of several options: using a personal identification number (PIN code), comparing signatures, *or* requiring additional ID. Here, the contribution type is *OR*, since any one of the offspring can be used to meet the parent. That is, Authentication can be accomplished by using a PIN code, by comparing signatures, *or* by requiring additional identification.

Catalogues also show possible decompositions of operationalizations into more specialized operationalizations. The catalogues also show the contributions of specialized operationalizations towards their parents.

AND and *OR* contributions are drawn with arcs, and the direction of contribution is towards the arcs. However the other contribution types (such as "+" in the figure) are drawn with arrowheads on the lines, showing the direction of *contributions* toward parent softgoals. Note that the direction of the arrows is typically the opposite of the sequence in which softgoals are generated. Contribution types affect how the graph is evaluated, and are explained in full in Chapter 3.

We use the term *decomposition* when the parent and offspring have the same kinds of softgoals. We have alread seen decomposition of NFR softgoals to other NFR softgoals, as well as decompositions of operationalizing softgoals into other operationalizing softgoals.

2.8 DEALING WITH IMPLICIT INTERDEPENDENCIES AMONG SOFTGOALS

At each step in the process, when we make choices in order to achieve a particular non-functional requirement (say, security of account information), it is very likely that some other non-functional requirements (e.g., user-friendly access, dealing with ease of use of the interface to a system) may be affected, either positively or negatively, at the same time. These interactions are very important because they have an impact on the decision process for achieving the other NFRs.

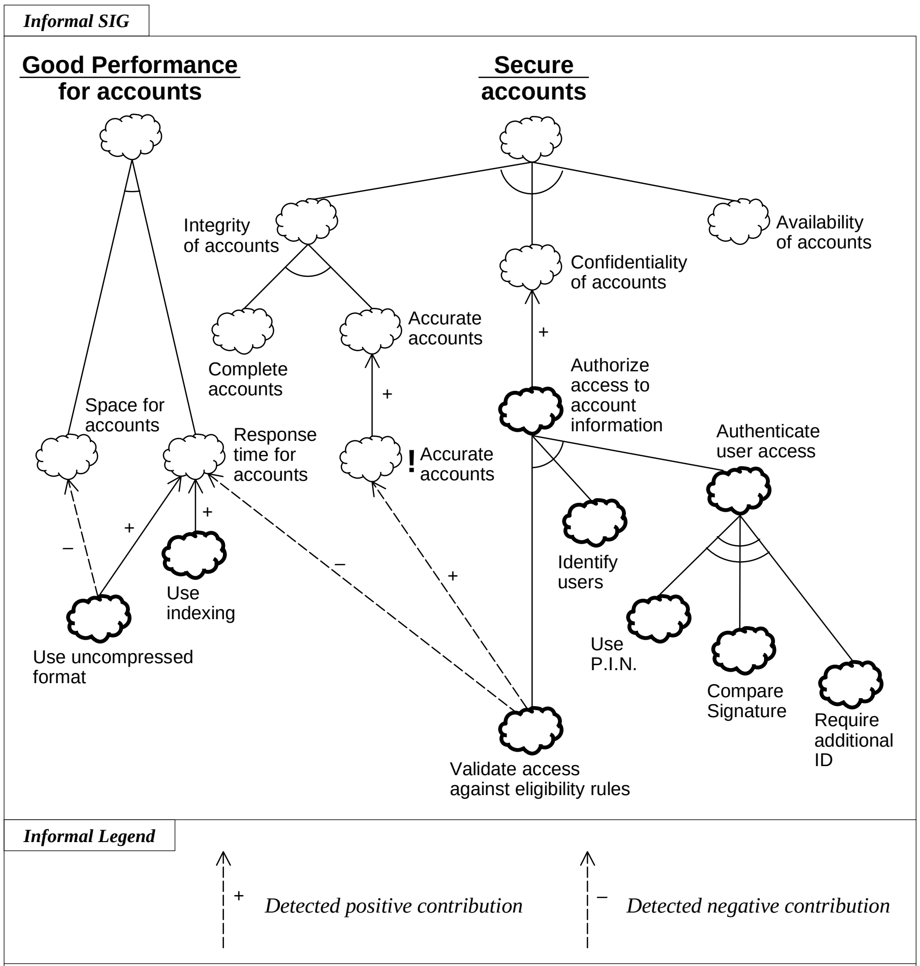
These interactions include positive and negative contributions. These different interactions can be dealt with in different ways.

We have already seen how developers can explicitly state interdependencies among softgoals, by using refinements. We call these *explicit interdependencies.* They are shown as solid lines in figures.

Now we consider interdependencies which are *detected* by comparing a portion of a SIG with a catalogue of relationships among softgoals. These *implicit interdependencies* (*correlations* among softgoals) are shown as dashed lines in figures.

Figure 2.9 shows some *correlations (implicit interdependencies)* among softgoals. These include positive and negative contributions.

Using an uncompressed format is negative (shown as "–") for space (because compressed formats are more space-efficient), but positive ("+") for response time (because we don't need to uncompress the data before processing it). The positive contribution was stated explicitly. Now the negative part of

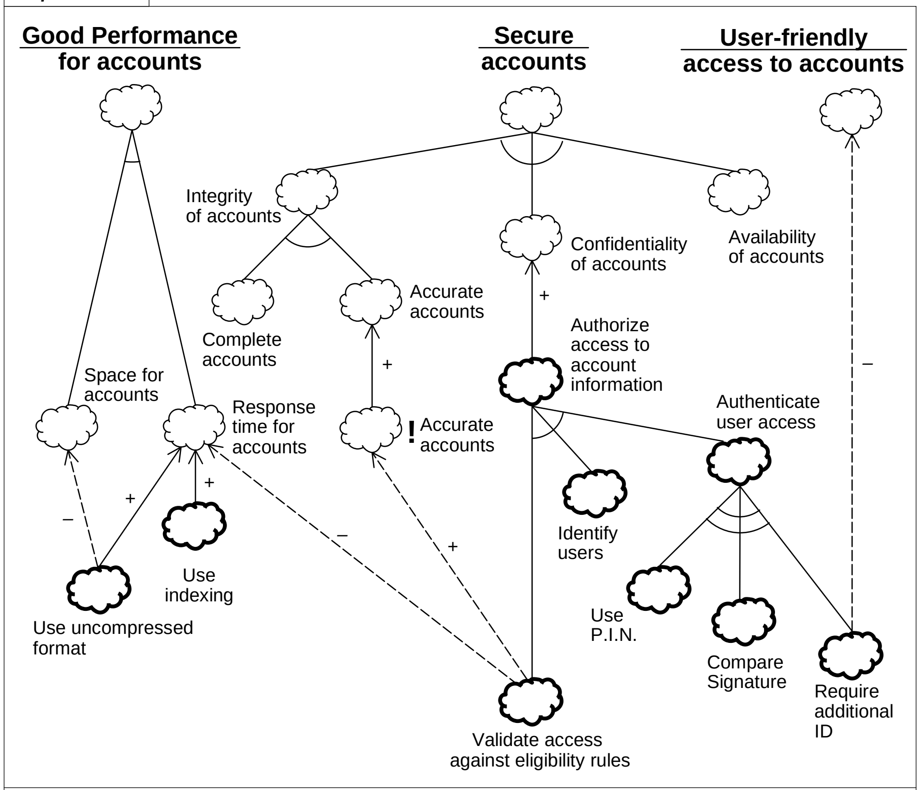


- ➤ Assistant automatically detects Accuracy-Confidentiality synergy (+).
- ➤ Assistant detects negative (-) impact of Validation on Response time.
- ➤ Assistant detects negative (-) impact of Uncompressed format on Space.

Figure 2.9. Detecting implicit interdependencies among existing softgoals.

this time-space tradeoff is detected using a correlation. By showing positive and negative contributions, correlations are one way of recording tradeoffs.

Informal SIG



- ➢ Assistant automatically detects Accuracy-Confidentiality synergy (+).
- ➢ Assistant detects negative (–) impact of Validation on Response time.
- ➢ Assistant detects negative (–) impact of Uncompressed format on Space.
- ➤ Assistant warns against possible omission of User-friendliness based on correlation rules

Figure 2.10. Detecting implicit interdependencies among existing and other softgoals.

As another example, Validate access against eligibility rules is one component of Authorize access to account information, which operationalizes the Confidentiality of accounts NFR softgoal. However, besides contributing to confidentiality, validation also happens to have a positive effect on the accuracy of accounts, since ill-intentioned users can be denied access and prevented from committing forgery. On the other hand, Validate access against eligibility rules has a negative contribution to Response time for accounts, since validation induces extra overhead.

Implicit interdependencies can be detected as the graph is being developed. This is done by consulting ("by hand," or with tool support) catalogues of positive and negative interdependencies among softgoals. These *correlation catalogues*, are discussed below in Section 2.12.

Figure 2.9 added *interdependency links* for correlations. Now we can consider correlations which add *softgoals.*

The examination of correlation catalogues may also lead to the identification of related NFRs which had not previously been considered relevant. In decomposing Authenticate user access, for example, one of the alternatives – Require additional ID – is detected to have a negative impact on User-friendly access to accounts. Although we had not been considering it in Figure 2.9, note that the NFR softgoal User-friendly access to accounts now appears in the SIG of Figure 2.10 as part of a correlation.

Thus correlations can add *softgoals* (Figure 2.10), as well as *interdependencies* (Figure 2.9) to softgoal interdependency graphs.

2.9 RECORDING DESIGN RATIONALE

An important premise of the Framework is that design decisions should be supported by well-justified arguments or *design rationale.* Reasons can be stated for making refinements, for selecting an alternative for the target system, etc. The Framework extends the goal-oriented approach to the treatment of arguments.

Let us consider two examples. First, we can state the reason for a *prioritization,* here of Accurate accounts. To support this prioritization, in Figure 2.11 we write Claim "Accuracy is vital."

We call this a *claim softgoal* (or *claim*). Claim softgoals are the third kind of softgoal. Earlier we saw NFR softgoals and operationalizing softgoals.

Here the claim is attached to an interdependency link, which connects the prioritized softgoal !Accurate accounts to its parent, Accurate accounts.

Note that we use the term *claim softgoal* to refer to the statement itself. When the statement is used to argue for or against something, that constitutes an *argument*, i.e., the argument refers to the *relationship* between the claim and the thing argued about. Claim softgoals are represented as dashed clouds in the softgoal interdependency graph. Their interdependency links represent the arguments.

As a second use of claims, we can rationalize *tradeoffs.* Here, the specialized operationalization Validate access against eligibility rules helps to achieve the more general operationalization, Authorize access to account information.

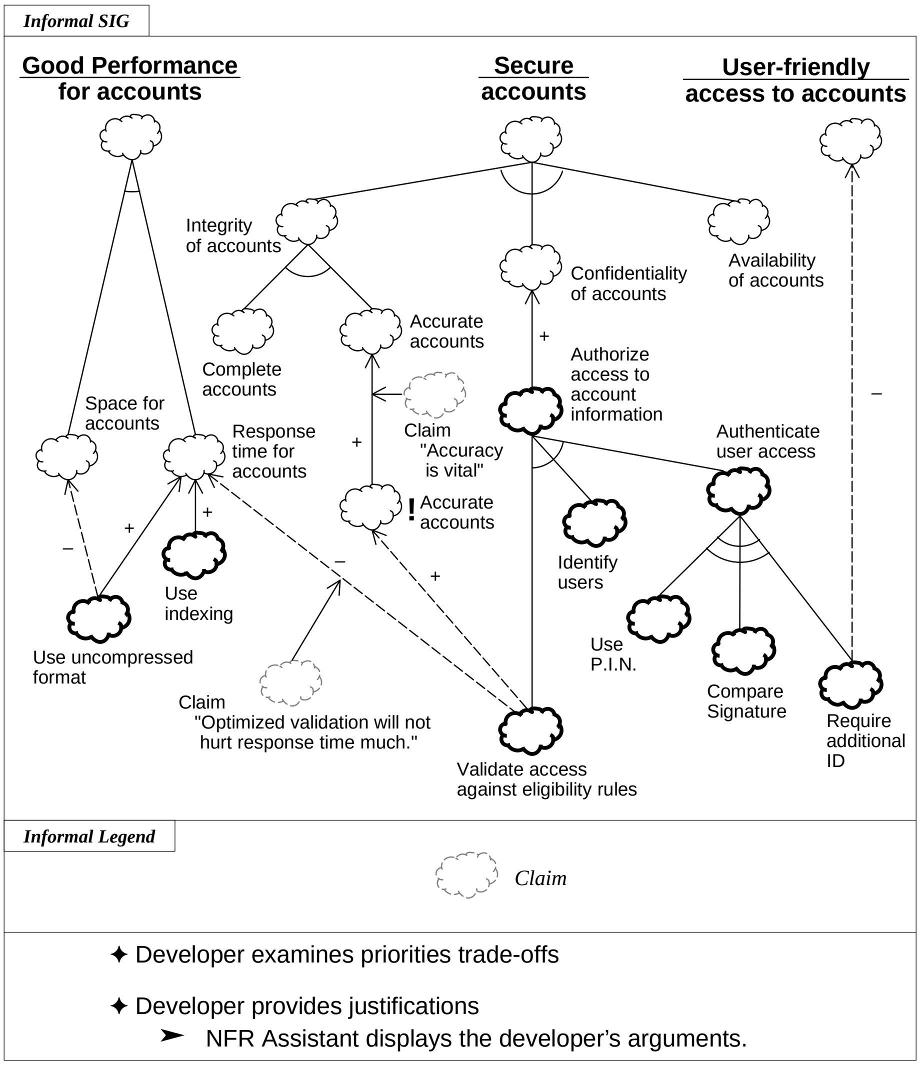


Figure 2.11. Recording design rationale.

At the same time, it has generated some correlations. It has a *positive* impact on the softgoal for accuracy of accounts, which is a priority. However, it also has a *negative* impact on the softgoal for good response time for accounts.

In this case, the developer weighs the tradeoff and feels that the Validate access against eligibility rules operationalizing softgoal is still worth considering, despite its negative contribution to Response time for accounts (Figure 2.11). In fact, this operationalizing softgoal will be chosen in the next section.

To support this position, the developer introduces the claim: Claim ["Optimized validation will not hurt response time much."] The claim notes that optimization of "Validate access against eligibility rules" will mitigate its degradation of Response time for accounts.

Rationale can draw on domain information, such as organizational priorities and organizational workload (e.g., the number of credit card sales per day), as well as development expertise.

Note that as these factors change, the reasons and decisions may change. For example, if the volume of daily sales rises dramatically, then priorities may change. These could be reflected in the rationale, and different operationalizations could be selected, which could have an impact on meeting the top softgoals.

Claims are treated as softgoals related by interdependencies and associated contributions, in the same way that NFRs and their operationalizations are. Claims make *contributions,* positive or negative, to other softgoals. Such contributions are not shown in the figures of this chapter, but will be discussed in the next chapter.

2.10 SELECTING AMONG ALTERNATIVES

The refinement process continues until the developer considers that the possible solutions for the target system are sufficiently detailed, and that no other alternatives need be considered.

Along the way, the developer has considered NFRs, domain information, and addressed ambiguities, priorities and tradeoffs. The developer has then considered possible operationalizations, design rationale, and interdependencies among softgoals, using the expertise from the developer and from catalogues. Of course, early decisions need to be considered and reconsidered when making later decisions. All this information is recorded in the graph, and is available to help the developer select among alternatives.

Thus, developers are aided in their decision-making, by referring to the graphs, which have a history of the design process.

In expanding the softgoal interdependency graph, the developer is elaborating on the possible subparts and alternatives that one would like to consider as means for achieving the initial high-level softgoals. The graph thus represents a design space over which design decisions are to be made.

As softgoals are being refined, the developer will eventually reach some softgoals which are sufficiently detailed. The developer can accept or reject them, as part of the target system.

Now it is time to choose from among the possible operationalizations, in order to produce a target system. In addition, appropriate design rationale should be selected.

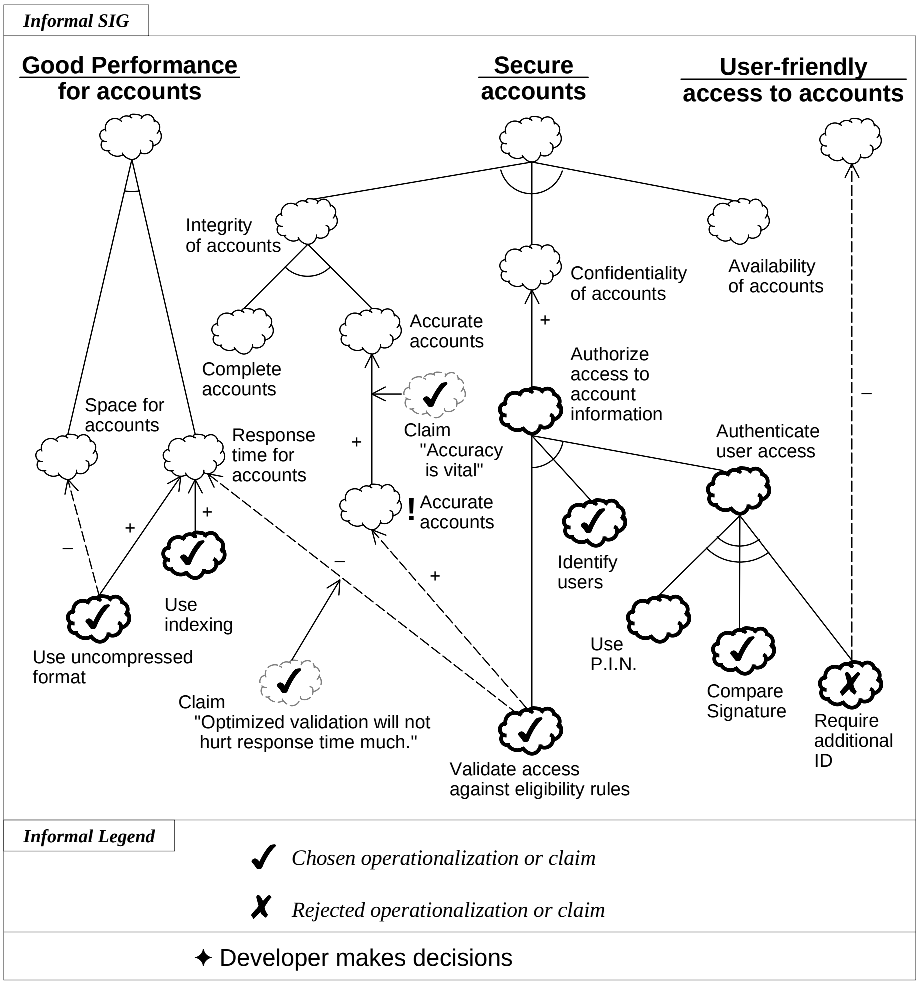


Figure 2.12. Selecting among alternatives.

There are choices of *operationalizing softgoals.* Of the many target alternatives in the graph (Figure 2.12), some are chosen (selected or "satisficed," indicated by "$\surd$") and others are rejected (denied, indicated by "$\times$").

Having identified three possible ways for authenticating user access, the developer decides that the Compare Signature operationalization is acceptable. Chosen solutions are represented in the graphical notation as a check-mark ("$\surd$") inside the node (Figure 2.12). On the other hand, rejected candidates, such as Require additional ID, are shown as "$\times$".

To aid confidentiality, Identify users and Validate access against eligibility rules are also selected. To aid response time, indexing and an uncompressed format are chosen. Note that a decision need not be made for every operationalizing softgoal. This is the case for Use P.I.N, which is left blank.

As is the case for operationalizing softgoals, *claim softgoals (claims)* are also either accepted (satisficed) or rejected (denied). The claim for prioritizing accuracy is accepted, hence a check-mark ("$\surd$") is placed in the Claim["Accuracy is vital."] claim softgoal. Likewise, the claim Claim["Optimized validation will not hurt response time much."] is accepted.

Now we turn to the impact of these decisions on top requirements.

2.11 EVALUATING THE IMPACT OF DECISIONS

The *evaluation* of softgoals and interdependencies determines the impact of decisions. This indicates whether high-level softgoals are met.

To determine the impact of decisions, both current and previous decisions are considered. Previous considerations and decisions are already reflected in the graph, as softgoals and interdependencies.

To reflect the nature of design reasoning, the evaluation and propagation of design decisions focusses on the question of whether a chosen alternative is "good enough," i.e., whether it meets a softgoal *sufficiently*. This style of reasoning is appropriate for dealing with non-functional requirements since meeting these requirements is often a matter of degree, not a binary true-false decision. The NFR Framework builds on the notion of *satisficing*, which was used by Herbert Simon [Simon81] to refer to finding solutions that are sufficiently good, even if they may not be optimal. To emphasize the difference between this style of goal-oriented reasoning from the more conventional, binary logic-based, goal-oriented reasoning (e.g., [Nilsson71]), we use the term *softgoal* to refer to the kind of goal that requires satisficing.

The evaluation process can be viewed as working bottom-up, starting with decisions, which are often leaves of the graph, and often at its bottom. Then the *evaluation procedure* (or *labelling algorithm*) works towards the top of the graph, determining the impact on higher-level main softgoals. These top softgoals reflect overall non-functional requirements as stated by the developer and people in the organization for which the system is being developed.

In the NFR Framework, the evaluation of softgoals is represented by assigning labels (such as "$\surd$" and "$\times$") to the clouds (softgoals) in the graph. Labels may be assigned by the developer , or computed from contributions from other nodes.

Roughly speaking, when there is a single offspring, a *positive* contribution "propagates" the offspring's label to the parent. Thus a satisficed offspring results in a satisficed parent, and a denied offspring results in a denied parent.

On the other hand, a *negative* contribution will take the offspring's label and "invert" it for the parent's label. That is, a satisficed offspring leads to a denied parent, and a denied offspring leads to a (somewhat) satisficed parent.

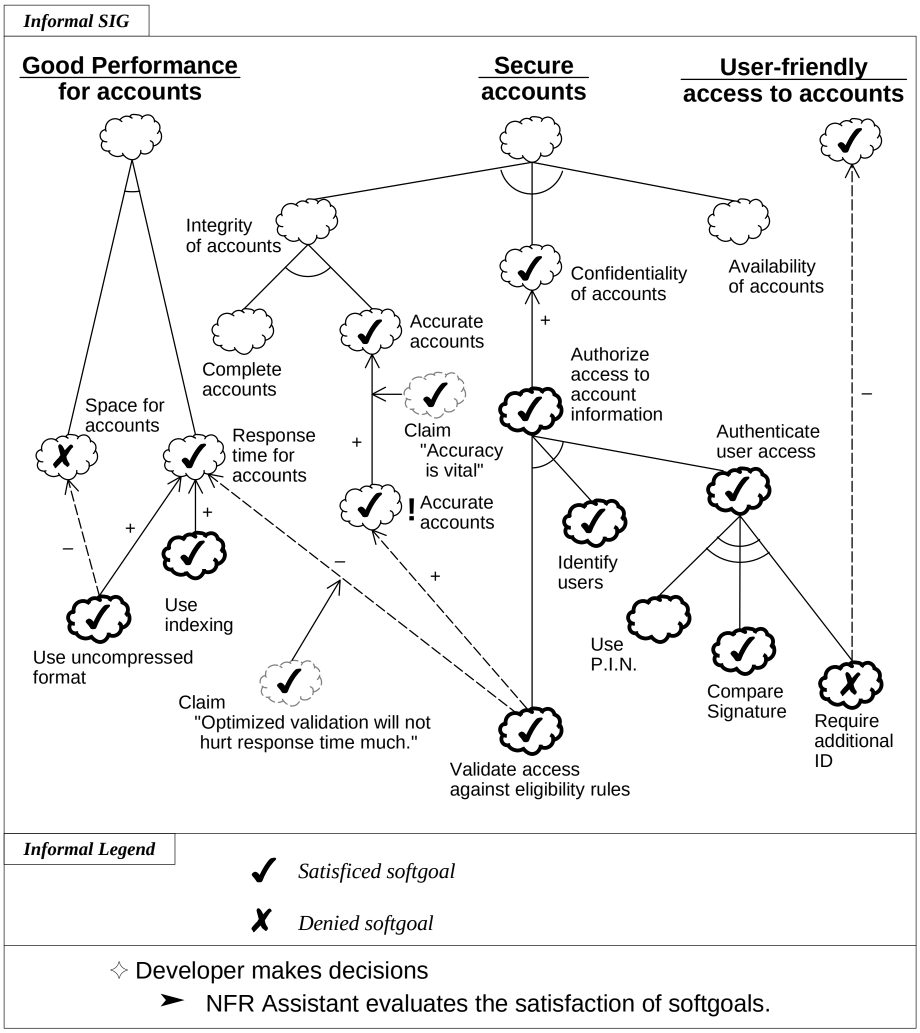


Figure 2.13. Evaluating the impact of decisions.

This is the case in the lower left of Figure 2.13. The operationalizing softgoal Use uncompressed format makes a negative contribution towards the NFR softgoal Space for accounts. In addition, Use uncompressed format is satisficed (the label is shown as "$\surd$"). When the satisficed label and negative contribution are considered, the result is that Space for accounts is *denied* ("$\times$").

Note that in Figure 2.12, "$\surd$" indicated "leaf" operationalizations or claims which were selected directly by the developer. Now in Figure 2.13 (and the remainder of this book), its meaning is made more general. It indicates softgoals which are determined to to be *satisficed;* this is determined by the developer, or by the evaluation procedure. Likewise "$\times$" represented rejected operationalizations or claims, but now more generally indicates *denied* softgoals.

Suppose a softgoal receives contributions from more than one offspring. Then the contribution of each offspring toward the parent is determined, using the above approach. The individual results are then combined.

For example, Response time for accounts has three offspring. Use compressed format is satisficed and makes a positive contribution, hence its individual result would be to satisfice the parent. The same is true for Use indexing. If these two were the only offspring, the combination of their individual positive results would lead to satisficing the parent. However, Validate access against eligibility rules is satisficed and makes a somewhat negative contribution to response time.

The developer combines the results — two satisficed and one denied — and sees there is a conflict. What value should be assigned to the parent softgoal? The developer could assign various values — satisficed, denied, or something in between. Here, the developer notes that claim that optimized validation will not hurt response time much, and labels Response time for accounts as satisficed.

Using the rule for satisficed softgoals and positive contribution links, !Accurate accounts is noted as being satisficed, and so is Accurate accounts.

It is interesting to note that the evaluation procedure works the same way, whether the interdependency link was explicitly stated by the developer, or implicitly detected.

Let us continue at the bottom right of Figure 2.13. Requiring additional identification is not chosen for authenticating access. This helps satisfice the requirement for user-friendly access to accounts.

Compare Signature and its siblings participate in an *OR* contribution to their parent, Authenticate user access. This means that if any of the offspring is acceptable, the parent will be acceptable. Compare Signature was chosen, so the operationalizing softgoal Authenticate user access can thus be automatically evaluated to be satisficed ("$\surd$").

Earlier, the softgoals Identify users and Validate access against eligibility rules were selected, hence satisficed. Since they and Authenticate user access are all satisficed, then a checkmark can be propagated "upwards" along the *AND* contribution to the Authorize access to account information softgoal. Then Confidentiality of accounts is satisficed.

Let us take stock of how well the target system would meet main non-functional requirements. Accuracy, confidentiality, response time and user-friendly access requirements have been satisficed. The space requirement has been denied. For other requirements, such as performance, integrity and security, we have not indicated if they are satisficed or denied. To obtain answers for some of these, we would need further refinement, further information, or a resolution of conflicts.

Interestingly, we have been able to address interactions between different kinds of non-functional requirements (here, accuracy and performance), even though the NFRs were initially stated as separate requirements.

In the NFR Framework, a softgoal can be assigned a *label.* So far, we have used labels such as "$\surd$" or "$\times$". In fact, there are actually more "shadings" of label values than these ones. For example, softgoals can be weakly satisficed, or weakly denied. The propagation of these labels along interdependency links depends on the type of contribution.

We have shown some contribution types, such as *AND* and *OR*, in this chapter. There are also other contribution types indicating various combinations of positive and negative, and partial and sufficient contributions. In addition, *claims* provide positive or negative contributions; these are omitted from the Informal SIGs in Figures 2.11 through 2.14. Full details of labels and contribution types are given in Chapter 3.

The propagation of labels is interactive, since human judgement is needed at various points. The evaluation procedure (whether executed by a developer "by hand" or using an automated tool) will propagate labels as far as it can, at which point the developer can step in to provide values as appropriate. At any time, the developer can override previously assigned or computed labels, and can change contribution types. Details of the evaluation procedure are given in Chapter 3.

Developers can assess the status of their designs at any time, by examining how the status of the most detailed decisions contribute towards the top-level softgoals that they started with. The developer can thus make informed tradeoffs among the available alternatives.

Relating Functional Requirements to Decisions and NFRs

We can also relate *functional requirements* to NFRs and the decisions made for the target system.

So far, graphs started with top NFRs and resulted in operationalizations being selected. Now we graphically relate them to functional requirements and their associated chosen target design *specification* or *implementation.*

The top of Figure 2.14 shows the functional requirements (for maintaining accounts) and the top level NFRs, relating to the security, performance, etc., of maintaining accounts. The bottom of the figure links the chosen operationalizations to a description of the target system (shown in a rectangle at the bottom right). The right side of the figure links this description of the target system to the (source) functional requirements (shown in an oval at the top right).

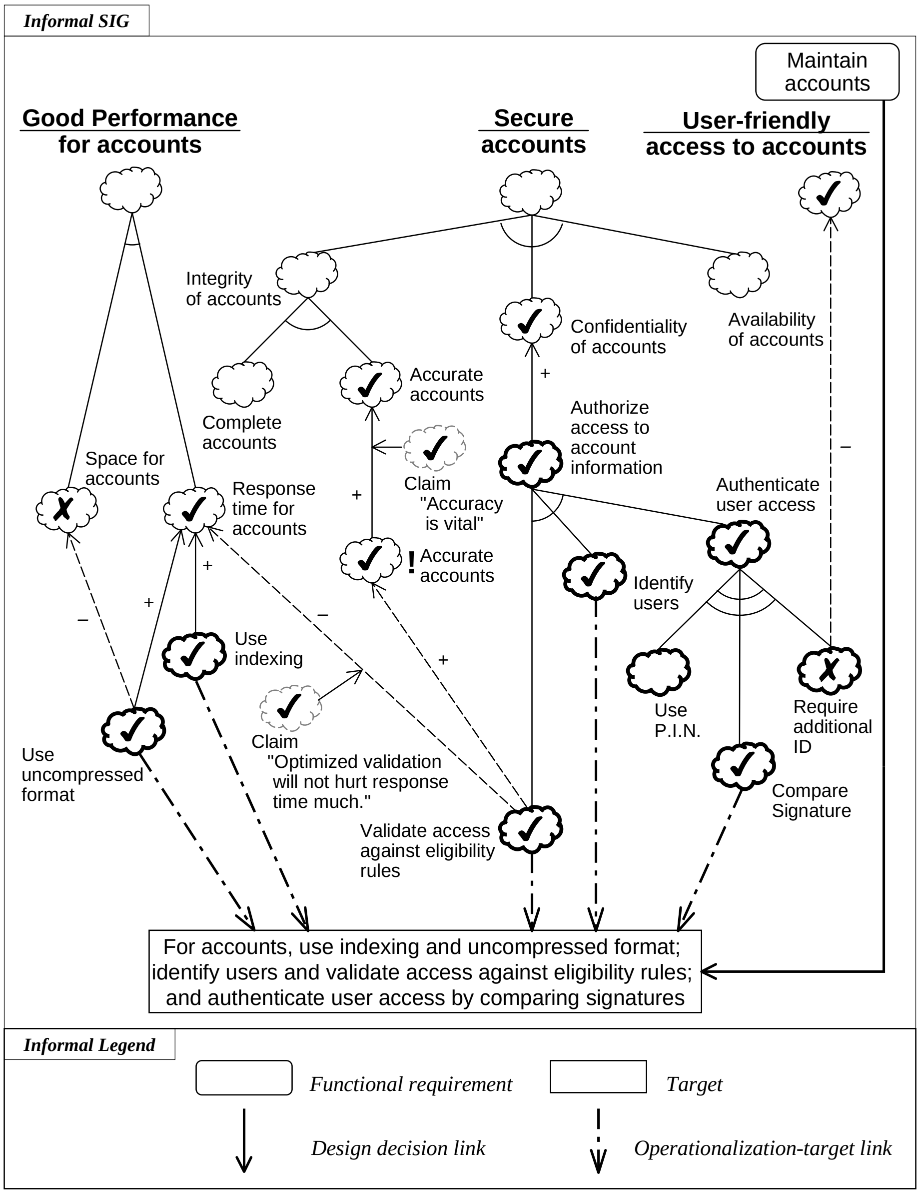


Figure 2.14. Relating decisions to Functional Requirements.

2.12 CATALOGUING DEVELOPMENT METHODS AND CORRELATIONS

Let us further consider how a developer comes up with refinements, including NFR decompositions, operationalizations, and claims that justify decisions.

In order to be able to bring relevant knowledge to the attention of the developer at each point in the design process, knowledge needs to be represented in a flexible catalogue structure. Three major kinds of catalogues are used to express design knowledge:

- *NFR type catalogues:* They encode concepts about particular types of NFRs, such as security and performance.
- *method catalogues:* They encode knowledge that helps refine graphs by decomposing softgoals and considering operationalizations.
- *correlation rule catalogues:* They encode knowledge that helps detect implicit interdependencies among softgoals.

Developers can browse through these catalogues to examine a current area of interest within a wide range of possible techniques.

Development knowledge can be catalogued to organize NFR types, methods, and correlation rules, which are used as a developer faces particular design decisions.

So far we have shown only a catalogue of NFR types in Figure 2.1. Let us now consider other kinds of catalogues.

Method Catalogues

Figure 2.15 shows a *catalogue of methods* for addressing the NFR of confidentiality. The catalogue is hierarchically classified: more specific methods (techniques) are placed under general ones.

The concept of *method* is applied uniformly to refinement throughout the Framework. Thus there are methods to assist in the decomposition of NFRs, methods for operationalizing, and methods for argumentation to identify claims.

Let us consider how catalogues can be used to aid refinement. Recall Figure 2.2, where we were starting with the NFR softgoal that accounts be maintained securely and with good performance. A search of catalogues (which can be done by hand, or be tool-assisted and semi-automated) for NFR types and methods could result in a suggestion to decompose an NFR softgoal into several sub-softgoals, which use sub-types of the type of the parent softgoal. This occurs through the invocation of a *method* which encodes this knowledge, here, about sub-types.

Some methods are domain-specific, e.g., dealing with particular development techniques and domains. Some other methods are fairly generic, applying to all softgoals of a particular NFR type. Others are quite generic, applying to a variety of domains and NFR types, e.g., a method decomposing an NFR

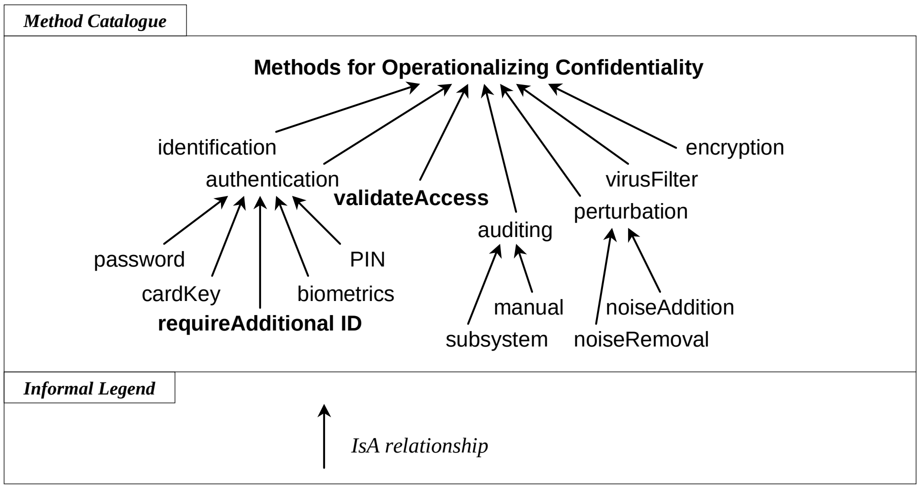


Figure 2.15. A catalogue of operationalization methods for achieving confidentiality.

softgoal for a data item into NFR softgoals for all components of that item. Methods that are generic tend to be more powerful, because they are applicable more widely. However, they can be harder to come up with. Many generic methods are already built into the Framework, and are developed by the Framework developers.

The NFR Framework comes with a collection of generic methods, which are described in Chapter 4. Other method catalogues are available for accuracy, security and performance requirements, and are discussed in Chapter 4 and Part II.

Note that the developer may have to select among a number of suggestions, possibly coming from different knowledge sources. The developer can also adapt a method by overriding some of its features.

Correlation Catalogues

Now let us consider a *catalogue of correlations,* showing *implicit interdependencies.* Figure 2.16 is an example of a catalogue of implicit interdependencies, and their contributions.

This catalogue shows the impact of various operationalizing softgoals (such as Validation and Indexing, shown on the vertical axis) upon NFR softgoals (such as Accuracy and Response Time, shown on the horizontal axis). The correlation catalogue entries show the contribution type. For example Validation makes a positive contribution towards Accuracy, and Indexing has a positive contribution towards Response Time.

Informal Correlation Catalogue

Impact of offspring *Operationalizing Softgoal*	**upon parent** *NFR Softgoal*				
	Accuracy	Confidentiality	Response Time	Space	User-friendliness
Validation	+	+	−		
Compression			−	+	
Indexing			+		
Authorization		+			
Additional ID		+			−

Figure 2.16. A catalogue showing the impact of operationalizing softgoals upon NFR softgoals.

Detected correlations are shown as dashed lines in graphs. In Figures 2.9 and 2.10, the implicit interdependencies between Use uncompressed format and Space for accounts, between Require Additional ID and User-Friendly Access, and among Validate Access against Eligibility Rules, Response Time, and Accuracy of Accounts are detected by correlation rules.

Catalogues can be prepared with different axes showing correlations between different groups of softgoals. For example, another correlation catalogue could show the impact of NFR softgoals upon other NFR softgoals. Yet another correlation catalogue could show the impact of operationalizing softgoals upon other operationalizing softgoals.

Catalogue-based correlations are detected by pattern-matching. Correlations can be used to establish interdependencies in a graph either automatically or after confirmation by a developer, depending on the developer's wishes. Like methods, correlations can be generic or specific.

2.13 DISCUSSION

In this chapter, we offered a glimpse into how the NFR Framework provides a systematic treatment of non-functional requirements, and how they are methodically introduced into steps in the design process.

During this process, the developer is constructing a record of how sub-softgoals contribute to higher softgoals, eventually contributing to the top-level softgoals. Throughout the development process, both selected and discarded alternatives form part of the development history, and the softgoal interdependency graph keeps track of the impact of decisions upon the top-level softgoals. These records are displayed graphically.

Catalogues of NFR knowledge help with the time-consuming, and often difficult, search for development techniques.

It can be possible, at least for small examples, to use the NFR Framework starting from "first principles," without the use of catalogues. However, our experience has been that catalogues are quite important, especially when considering NFRs for larger systems.

While it can take some time to develop methods and catalogues, our experience is that this results in a payoff, by speeding up the refinement process. Chapter 4 presents generic catalogues of methods and correlations, while Part II presents catalogues for particular types of NFRs (accuracy, security and performance).

Literature Notes

This chapter is adapted from [Chung94a,b].

The goal-oriented approach we have taken is meant to be intuitive, following and supporting the developer's reasoning. However, it is not the same as the approaches in artificial intelligence or automated reasoning, such as AND-OR trees [Nilsson71]. By taking a "satisficing" approach [Simon81], the aim is to offer flexible support for a developer's reasoning, and to allow it to be recorded. The purpose is not to automate development. Instead, development is intended to be highly interactive, where the human developer is in control.

The catalogues of refinement techniques and tradeoffs (e.g., [Chung93a] [Nixon97a] [Yu94b]) are based on work done by researchers and practitioners in particular areas. These include security [ITSEC91, Parker91, Clark87, Martin73], accuracy [Pfleeger89], and performance [C. Smith90, Hyslop91], as well as information system development. In addition, catalogues can deal with other NFRs, other classes of issues, e.g., Business Process Reengineering [Hammer95], and other aspects of development, e.g., software architecture [Garlan93].