17. Reusable Rationale Blocks: Improving Quality and Efficiency of Design Choices

Wiebe Hordijk[1]*, Roel Wieringa*

Abstract: In the current practice of designing software for user organizations, as experienced by the authors, designers often produce design knowledge again and again for every decision: They re-invent the wheel. We want to improve the quality, predictability and efficiency of the software design process by reusing design knowledge. Our proposed solution consists of Reusable Rationale Blocks (RRBs). An RRB is a schema and a notation to write down decision rationale. To manage RRBs, we introduce a generalized design space, that consists of a collection of RRBs. And to use RRBs, we define a process that can be added to any design process, as well as a set of heuristics to be used in applying this process. We illustrate our solution by a few examples taken from our own experience.

Keywords: Architecture, Design, Rationale Management, Reuse, Software Process Improvement

17.1. Introduction

This chapter proposes an improvement of the process of designing software for user organizations. It is based on our experience in designing enterprise information systems, but our solution is stated in general terms applicable to any software design process for user organizations.

17.1.1. Problem

In current practice the use and production of design knowledge by practitioners does not lead to a growth of design knowledge in the community of practitioners. Designers re-invent the wheel repeatedly [7] [Chap 1., Sect. 1.4.4.3 in this book]. The same knowledge is produced again and again, and is not reused for later, similar decisions. This means that design takes more effort from the designers than needed. It also entails the risk of lower quality of results, and because knowledge is not reused, it makes the properties of the final product hard to predict, loading a lot of risk on the project.

[1] Partially sponsored by Ordina BV.

17.1.2. Goal

As stated, the goal of our approach is to improve the quality [7] [Chap. 1, Sect. 1.4.3 in this book], predictability and efficiency of the software design process by reusing design knowledge. We take a *generalized approach* to reuse [7] [Chap. 1, Sect. 1.4.4.3 in this book]. Decisions can be made more repeatable by reusing information used in them in later, similar decisions, and their quality can be improved by recording their impact on software quality. To achieve our goal of reuse, we provide a structure and a process for such reuse, and we produce and validate the design knowledge to be reused.

Before we proceed, we need to clarify this goal in a number of ways. First, our goal is not to provide a general knowledge management tool for reuse of design knowledge. We focus on reuse of decision rationale. This is knowledge about which options there are for a design decision, and how each option affects the relevant requirements. It is useful to reuse other knowledge too, but this is not what we do here.

Second, the application of our solution is in the areas of Engineering of systems and Acquisition of system components and infrastructural components [7] [Chap. 1, Sect. 1.6.6 in this book]. We don't rule out that the approach could be used in other process areas, but we have not looked at this.

Third, our experience is limited to Enterprise Information Systems (EISs), which are systems that serve administrative purposes in organizations and are used by people in those organizations. EISs typically store and use data, have some business logic and one or more user interfaces. But the solutions that we come up with are stated in terms applicable to any software design problem.

Fourth, we only consider custom-built applications, as opposed to off-the-shelf software. These applications are of course built upon commercially available components.

Fifth, we focus on higher-level design decisions, and not on the lower-level ones. The higher-level decisions cost more effort and represent more risks than lower-level decisions. Therefore, there is a higher demand for quality of higher-level decisions than there is for lower-level decisions. In our approach, we focus on the higher-level decisions. Also, these involve teams of people from different parts of an organization, and should be documented well for accountability and maintainability.

17.1.3. Solution outline

Our proposed solution consists of four parts:

- Reusable Rationale Blocks (RRBs), a schema and a notation to write down decision rationale.
- A generalized design space, consisting of a collection of RRBs for EISs.
- A process description, where the use of RRBs is added to a standard design process. This description should enable you to add the use of RRBs to your own design process. The process also tells how to create RRBs.
- A set of heuristics to be used in applying the RRB process.

The RRBs are the core concepts of our approach. They are generalized pieces of decision rationale. They each contain one question, a set of solution options to the question, and an evaluation table where the options are compared to each other with respect to a set of requirements, called criteria. This is similar to the Questions-Options-Criteria (QOC) approach [13]. As criteria we use quality indicators, taken from the extended ISO 9126 quality model [16] because they are measurable and of interest to the customer. See Sect. 17.1.4.2 for a comparison between QOC and our approach.

The generalized design space is a set of RRBs on a web site (http://is.cs.utwente.nl/QUIDS/). It is called a generalized design space because a regular design space is about a concrete system, while a generalized design space is about a class of systems. We will just write “design space” instead of “generalized design space”. In addition to the decision rationale in the RRBs, the design space adds a ‘super/sub-problem’ structure, giving designers a minimum checklist of decisions they should have made before the design can be ready.

The process description is our idea of how RRBs should be used in practice, both in systems design and in research, but focusing on design. We don’t prescribe a particular design process, but rather show how a design process can be extended to use RRBs. Our solution thus is prescriptive [7] [Chap. 1, Sect. 1.2.2.1 in this book].

Our solution is highly intrusive [7] [Chap. 1, Sect. 1.2.2.2 in this book] on one hand, because the design process is changed. On the other hand, we argue that a good designer naturally documents the argumentation behind design decisions at the same time the decisions are made, and that doing this in our notation does not add extra effort. This is also proposed in [7] [Chap. 1, Sect. 1.5.2 in this book]. In our opinion, decisions should be first-class citizens of software architecture. Under those assumptions, our approach is non-intrusive. In other words, documenting rationale in such a way that it can be reused later shouldn’t take more effort than just documenting the rationale.

17.1.4. Related work

This section describes the differences between our solution and some alternative approaches from rationale management and architecture research.

17.1.4.1.Design Patterns

An important approach to the reuse of design knowledge is the use of patterns, which are descriptions of problems and corresponding solutions according to some format [1,6,8,9]. Design Patterns describe solutions in detail, which makes them ideal for teaching, knowledge management, and for describing *how* a system is structured. One problem with patterns is that they do not provide guidelines about how to choose among several patterns that all in different ways satisfy a set of requirements. As mentioned in [7] [Chap. 20 in this book], one pattern describes rationale for one solution instead of several ones. In the terminology of Lee and Lai [12], patterns contain *design rationale* (they tell us which problem a solution solves), but no *decision rationale* (they do not tell us how to choose among patterns). RRBs, on the other hand, describe alternatives and effects, but do not explain the options in great detail. They are meant for making and justifying design decisions. Ideally, the options in an RRB are described as patterns, or by references to known patterns, so that design rationale and explanation are in one place.

17.1.4.2.QOC

The Question-Options-Criteria approach was introduced [13] to illustrate and analyze the arguments that lead to design decisions about individual systems in a graphical way. Each option can score in only two ways on a criterion: good or bad. Our approach generalizes this to generic design problems and refines the scoring of options. Our design problems are QOC's Questions, our options are their Options, and our quality indicators are a refinement of their Criteria. We use the term ‘design problem’ because we think that the term ‘question’ is too broad.

We refined QOC by restricting ourselves to measurable quality indicators, and by ranking options on each quality indicator on more than two ranks. We felt we needed more ranks than two for codifying general design knowledge.

Another difference is in the goals of the methods. QOC is a descriptive method, aimed at describing the rationale of decisions. Our approach is prescriptive, aimed at improving design processes, for which we use rationale as a means to an end.

17.1.4.3.ADD

Attribute Driven Design [5] is a field of research at the SEI. Their unit of research is the Design Primitive (also known as Architectural Mechanisms). An example of a design primitive is caching. Design primitives can be compared to our Solution Options, except that a Design Primitive is not linked to a Design Problem. Design primitives are problem-oriented: “The performance is too low, so we use caching to fix that.” RRBs are choice-oriented: “Should we use caching or not?”

17.1.5. Structure of this chapter

We start with a more thorough explanation of RRBs and the design space in Sect. 17.2. Then we show how to use RRBs in Sect. 17.3 about the RRB Process. In Sect. 17.4, we illustrate the process in three cases from a system design project in which one of the authors took part as an architect. In Sect. 17.5, we present the lessons learned from the cases, which make up the heuristics of our solution.

17.2. Reusable Rationale Blocks and the Design Space

This section first explains how design decisions and software quality are related, and then explains how such knowledge is documented in RRBs. We start our explanation with an example of an RRB, after which we show the general structure of RRBs. Then we proceed from individual RRBs to our design space.

17.2.1. Design Decisions and Software Quality

Central to our approach is the assumption that decisions taken during the design of a system partly define the quality that the system will have after implementation. The quality of a system falls apart in many quality attributes, which can be measured using quality indicators.

17.2.2. RRB Example

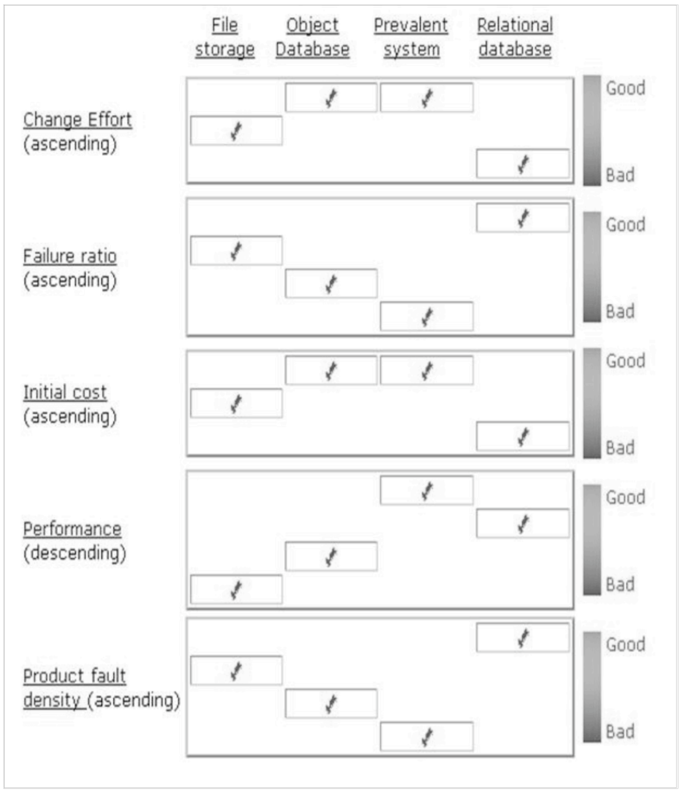

Fig. 17.1. Web site screen shot of Reusable Rationale Block 'Data storage type'. See http://is.cs.utwente.nl/QUIDS. Prevalent systems are a new and experimental kind of systems that keep all data in memory; see http://www.prevayler.org/wiki.jsp for more information.

Fig. 17.1 shows (part of) the RRB of the design problem "data storage type". The leftmost column of the RRB shows the quality indicators that are affected by the solution options. The rows show how each solution option is ranked by each quality indicator. For example, the option "Relational database" scores best on the quality indicator "Failure ratio". This means that in a system where data are stored in a relational database, we expect the failure ratio will be lower than with other types of data storage. Note that a single column when taken out of the table does not have meaning: the RRB does not say that "Relational database" scores better on "Failure ratio" than it scores on "Initial cost".

The orderings in a row are partial. When two options have equal ranks for a quality indicator, such as Object databases and Prevalent systems for "Change effort", we do not say that those options are indifferent with respect to that indicator, but rather that they are indistinguishable. The statements we are making with respect to "Change effort" is that option 1 is worse than options 2 and 3, and option 4 is worse than all the other options.

Each row in the table is a hypothesis about the effects of options on the system's quality. In our design space, we motivate these hypotheses by reference to the literature [6,9,11], as well as our own experience.

17.2.3. RRB General Structure

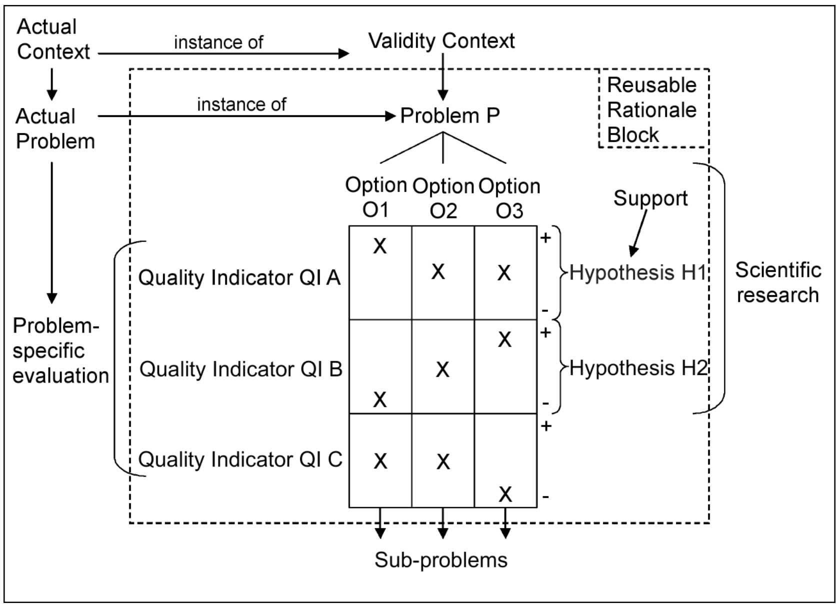

Fig. 17.2. Schematic representation of a Reusable Rationale Block in its environment. Some terminology is shortened because of space limitations.

The general structure of a Reusable Rationale Block (RRB) is shown in Fig. 17.2. A design problem is described in conjunction with its available solution options and their effects. It occurs in the context of an earlier design problem, and provides the context for further design problems. In the RRB, we describe this in a general way, but in an actual piece of software we will encounter an actual design problem in an actual context. The RRB, on the other hand, only describes a generic problem, existing in a context described generically, called a validity context. The designer must relate the actual context to the validity context, and the actual problem to the generic design problem described in the design space. Then the designer can make his or her evaluation of the options based on knowledge of the actual problem in its actual context. This is shown at the left-hand side of Fig. 17.2. At the right-hand side, we show that every row in the table is a hypothesis that can be investigated by further empirical research. In future research, we will try to validate or refute some of the hypotheses in our design space empirically. The structure of a design problem with its context can be reused in other situations, provided the design problem and context match the specific situation.

17.2.4. Design Space

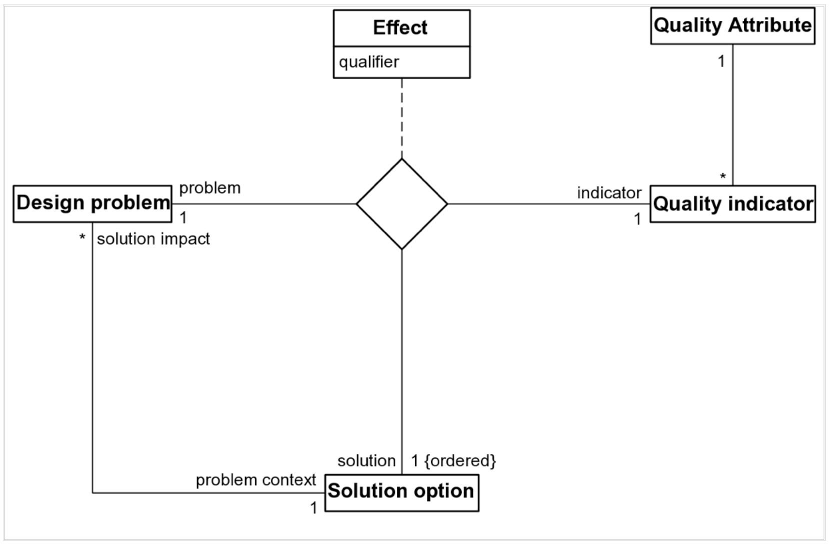

Fig. 17.3. UML Static structure diagram of the design space.

As said earlier, an RRB contains knowledge about a particular problem in the design of systems, their solution options, and the relations between the problems and solutions. Whenever a design problem is present, the corresponding RRB documents its different options and their different effects on the system's quality attributes. This can be modeled as a ternary relation called *Effect* between a design problem, solution option and quality indicator (Fig. 17.3).

To link multiple RRBs together into a design space, a solution option can be the problem context for more detailed design problems. For example, after choosing the option “relational database” for the design problem “where should the data be stored”, the design problem “what should the data model be?” is created, which also has a corresponding RRB in the design space. The design space therefore has the structure of a tree.

17.3. RRB Process

The RRB process is an approach to software design which uses decision rationale to improve design quality, predictability and efficiency by reusing design knowledge. In this approach, rationale is a necessary component of the design, not a by-product for later use. Decisions are first-class citizens of the design world, at least as important as classes, components and subsystems. Since design is a decision-making activity, documenting decision rationale should be as natural to a good designer as documenting source code is to a good programmer.

In this section, we propose how to use RRBs in practice and in research. In Fig. 17.4 we show a process model consisting of the design cycle on the right-hand side, the research cycle on the left-hand side, and flows between them, making a double cycle.

The basic design cycle is shown on the far right of Fig. 17.4 [14]. In the *Problem analysis* phase, the designer develops an understanding of the problem, and identifies criteria by which the solution will be evaluated. Here the Design problem and the Quality indicators of Fig. 17.3 are generated. In the *Solution synthesis* phase, possible solution options are generated in some way. Sources for solution alternatives include the designer’s experience, text books and the Internet. The quality attributes of these alternatives are predicted next. This *property prediction* can be done for example by comparing solution alternatives to known existing systems, by modeling the solution, or by prototyping. Here the Effect relationships of Fig. 17.3 are established. The properties can be presented in the form of a rationale table such as in Fig. 17.1 or the tables in Sect. 17.4. Then the predicted properties are evaluated against criteria (*solution evaluation*). This leads to either a choice for one of the alternatives, or a decision to synthesize new solutions or analyze the problem more thoroughly.

We extended this basic design cycle with a *Problem matching* activity, in which the designer uses RRBs. *Problem matching* influences some of the other activities. In *Problem matching*, the designer tries to find an RRB which matches the actual problem at hand. This means that the actual problem can be regarded as an instance of the RRBs problem, and the context of the actual problem as an instance of the RRBs validity context. When a matching RRB is found, this makes the rest of the design cycle much easier. The RRBs *options* are input to the *Solution synthesis* activity, and the effects that the options have on quality indicators are input to *Property prediction*. When *Problem matching* does not yield a matching RRB, the regular design cycle activities are followed.

The designer must be aware that the RRB is more general than the actual problem, so in the actual problem, other options and effects may apply. That is why *Problem matching* does not replace the other activities.

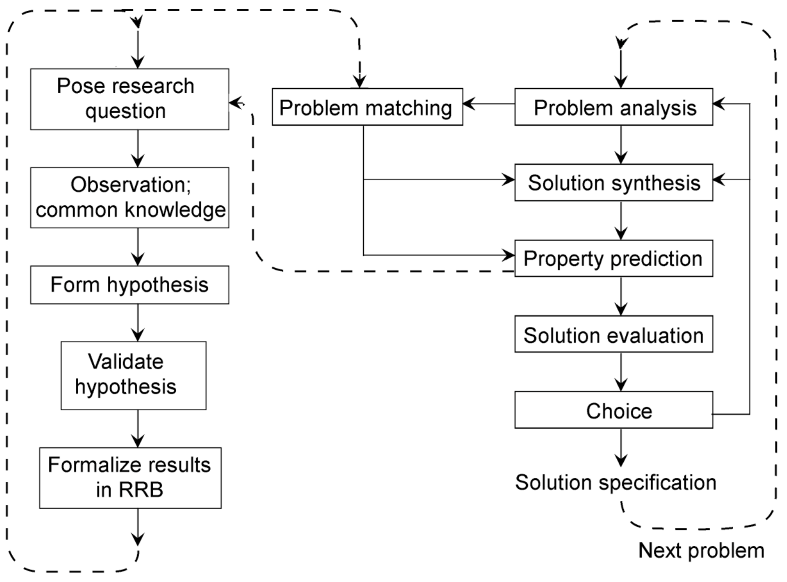

Fig. 17.4. Process diagram of the design cycle and research cycle linked together.

On the far left of Fig. 17.4 we find the research cycle, assuming that the research hypotheses are stated in the form of RRBs. This research process can be used in an academic context to yield knowledge applicable to a class of problems, as well as in an organization that needs the knowledge to solve a particular problem.

Finally, Fig. 17.4 shows that the design cycle and research cycle are linked together. Note that the RRB process has many similarities with the more generic Experience Factory approach [3]. Our approach is more specific to design rationale. An RRB can be seen as a way to package experience, in the Experience Factory terminology.

17.4. Illustrations

To make the RRBs at http://is.cs.utwente.nl/QUIDS practically usable, we provided a Word template. Nothing more sophisticated is needed at this point in time. We use this template in the examples below. Note that in the Word template, we use pluses and minuses rather than vertical positioning of crosses to indicate ranking. This is convenient in Word, but it has the danger that designers may believe that it is meaningful to add and subtract pluses and minuses. We applied the current version of the design space in an industry project at a large Dutch government body. We investigated the application of the theory by using action research, a research method in which the researcher is actively involved in the activities that are being investigated [2,4]. To improve understandability of the examples, we simplify the system and the design decisions.

In these illustrations, all decisions are product choices. This is caused by the state of the project these examples have been taken from. We believe that other design decisions, like which pattern to use for a certain part of the software, can use the same process. To validate this is part of future research.

17.4.1. Setting

These cases took place in a large Dutch government body. The system under design (SuD) was an administrative system that supported one of the organization's most important primary processes. A distributed architecture had been chosen for the system, with a hub-and-spoke layout with one central node and 150 remote nodes, and asynchronous messaging as communication means between the nodes.

The rationale for choosing asynchronous messaging was that some of the communication channels between the nodes had too high latency for the system to perform well enough with synchronous communication. The hub-and-spoke layout was given by the geographical distribution of the end users. The organization had already chosen a programming language and an application server.

Up to this point, the architecture was given. These descriptions formed the context of subsequent design problems, and those are the subjects of the case studies in this section.

The team designing the system's architecture consisted of one project leader who was also a domain expert, one software architect who was also a domain expert, and one software architect (Hordijk). The team met weekly with the "architecture board", a group of representatives from the organization's IT infrastructure department, technological policy makers and the application development team leader. The time frame of the design efforts was from November 2004 to April 2005, when implementation started.

17.4.2. Case Description Format

Each case presented below is described according to the following format.

- Introduction, describing the design problem that was solved in the case and the interest of the case to the chapter.
- Narrative, describing the events that happened in relation to the case in chronological order. The narrative consists of paragraphs named according to the steps in the design cycle, with an extra level for iterations in the first case. Each step can appear more than once because of the iterative nature of design.
- Lessons learned from the case.

17.4.3. Case 1: Message-oriented middleware

The most important decision for this system was which protocols and products to use for communication and message routing between the nodes of the system. For this decision, no matching RRB was found, so the regular design cycle was followed. We needed five iterations through the design cycle before arriving at a choice. We give no narrative here due to space limitations. The most important lessons learned are in Sect. 17.5.

17.4.4. Case 2: Remote Data Storage

This case illustrates how to use an RRB in an actual design decision, and that designers must still use their own knowledge and common sense when applying an RRB.

The next-most important design decision in our project was which kind of data storage to use for storing data on the remote nodes of the system. This decision limited the possible options for some other decisions, so it had to be answered first.

Problem analysis. Problem analysis for this decision raised interesting questions, which we answered in a meeting with domain experts. We needed to know what data were stored remotely and why. It turned out that two kinds of data were stored remotely. The first kind was data that was entered remotely, which was stored there for future reference but also sent to the central system. The second kind of data was entered in the central node, sent out to the remote nodes and stored there, in a cache, for faster reference. So all data in a remote data store either came out of, or would soon be sent to, the central database. This made reliability and backup facilities less important for the remote data stores.

Problem matching. After problem analysis, we found an RRB that matched our problem, shown in Fig. 17.1. Because this RRB was a bit limited and did not include any specific products, but only product types, we decided to generate extra options and quality indicators to be complete.

Problem analysis for criteria and Solution synthesis. Options and quality indicators for this decision were generated in one brainstorm meeting with the architecture board. The brainstorm added new options and quality indicators to the RRB in Fig. 17.1. The options ‘Object Database’ and ‘Prevalent system’ were intentionally left out, because they did not score well enough on ‘Product fault density’, which is an indicator for Maturity. No matter how good these products may be, they were not yet widely in use, and our organization did not want to be a technological fore-runner. This ended the solution synthesis activity. The RRB was used further in property prediction.

Property prediction. The property prediction of the options in the RRB in Fig. 17.1 and our actual problem differ on some quality indicators. In the actual problem, three different RDBMS products were considered, with quite different quality indicator values. For example, we even regarded a particular RDBMS as less mature (indicated by scoring lower on product fault density) than file storage, while in the RRB, RDBMSs are considered more mature. Also, in the actual problem we added some quality indicators that were relevant in the specific context. This illustrates that an RRB contains generalized design knowledge, which should be tailored when used for actual problems. Still, RRBs are useful because they structure the decision and present an initial knowledge base to start from.

	Comma-separated files	XML files
Failure ratio	–	+
Initial cost	–	+
Disk space usage	+	–

Table 17.1 Rationale for discarding the option Comma-separated files. Disk space usage was added as a criterium in the brain storm session.

To make property prediction more efficient, we tried to reduce the number of options as quickly as possible. The option ‘comma-separated files’ could be compared locally to the option ‘XML-files’, as shown in Table 17.1. This shows that between the two, XML-files were better than comma-separated files, so we could discard comma-separated files. Likewise, the alternative RDBMS Product B was compared to Product A, and the only differences were that A had better Resource usage and B had better Performance. Resource usage was more important at the remote nodes and Product A would probably meet the minimum performance requirements, so Product B was discarded. Now we had four options left, and they were evaluated against all our criteria in Table 17.2. For this evaluation, we used the information in the RRB in Fig. 17.1, supplemented with product documentation and market knowledge.

	XML-files	RDMS ‘lite’	Open source RDBMS	RDBMS Product A
Change effort	++	–	–	+
Failure ratio	–	+	+	++
Initial cost	++	+/–	+	++
Performance	–	+	+	+
Product fault density	+	+	–	++
Maintenance effort	+/–	+	–	+
Installability	+	+	?	–
Resource efficiency	++	+	+	–
Vendor lock–in	++	–	+	–

Table 17.2 Rationale table for remote data storage type. The last three criteria were added in the brain storm session.

Solution evaluation and Choice. The criteria Product fault density, Failure ratio and Initial cost were of high importance to the organization. Installability, Resource efficiency and Vendor lock-in were of minor importance. Therefore, RDBMS product A was chosen.

Lessons learned. This case illustrates how the knowledge contained in an RRB can be used in decision making. We learned the following lessons from it.

- It shows that matching the RRB to the actual problem is a difficult step, because many factors are involved that influence which options are available, which quality indicators are important, and what effects the options have. One should therefore take these factors into account and tailor the RRB to the problem at hand.

- The case also shows that designers should deal with the decision making process in an opportunistic way, cutting out options as early as possible when they can show that an option is inferior to another in small iterations of the design cycle. This opportunism saves time, but still preserves the accountability and reusability of the rationale.
- Another thing we learned from this case is that it was invaluable to have access to people who know the local infrastructure and the products that run on it. This local knowledge may continue to be more valuable than the general knowledge stored in RRBs.

17.4.5. Case 3: Central Data Storage

This case shows that application of the same RRB in different contexts can lead to different design decisions. Another important decision in our project was which kind of data storage to use at the central node of the system. For this decision, the rationale table constructed in the second case was reused. The quality indicators that were important for the central node were different from those at the remote nodes, which led to a choice for the same product but for different reasons.

Problem analysis. The quality indicators Maintenance effort, Installability and Resource efficiency were less important for the central data store than for the remote ones, because it was only one node instead of 150 and central resources could easily be scaled up. Failure ratio and Product fault density, however, were far more important, because the central database double-served as a backup for the remote data stores. Also, some data was stored in the central database only.

Problem matching, Solution synthesis and Property prediction. From the original RRB in Fig. 17.1, we saw that according to our quality indicators, an RDBMS was the only viable option. Because of Failure ratio and Product fault density, we only wanted to consider the RDBMS products that were already in use in the organization. Now we still had the two RDBMS products A and B that the organization already used as options. These were evaluated against each other in Table 17.3. Table 17.3 was derived from Table 17.2 by removing the options that we could see were not viable, and adding RDBMS product B back in. RDBMS A and B are evaluated against each other.

	RDBMS product A	RDBMS product B
Change effort	+	–
Failure ratio	++	++
Initial cost	++	+
Performance	+	+
Product fault density	++	++
Vendor lock-in	–	–

Table 17.3 Rationale table for central data storage product.

Solution evaluation and Choice. This evaluation was presented to the architecture board, and RDBMS product A was chosen unanimously because everyone could easily see that the difference in initial cost and change effort made product A the better choice.

Lessons learned. This case shows that the same RRB applied to different design problems can yield the same result for different reasons. It did because the context of the actual problem was different. This case also shows that it is useful to reuse design knowledge, as the same knowledge in the RRB was used in two cases, and the rationale table from the previous case was reused in this one.

17.5. Discussion and conclusions

17.5.1. Lessons learned

Iterations through the design cycle do not always consist of sequences of steps through the entire cycle followed by a jump back to the start. Some iterations do, but others consist of backtracking to an earlier stage to get more information, and still others consist in jumping forward to later tasks before proceeding with the current one. In general, though, earlier iterations focus on earlier design tasks and later iterations focus on later tasks. This agrees with an observation made by Witte in a massive empirical survey of decision processes [15].

An existing RRB was used in case 2, and it was extended with quite some extra options and criteria. The results were fed back into the RRB and reused in case 3. With a less complex problem and most of the knowledge already there, this decision was comparably easy.

We have found many cases in which one problem affects another problem. In those cases, we keep the problems apart, and first decide on the affecting problem, and later on the affected problem. This ‘divide and conquer’ tactic keeps the evaluation tables small and the decisions easy.

We have seen that RRBs give most of their input early in decision processes. They give the decision process a ‘quick start’ by providing crude versions of the options and their effects for typical design problems. The current body of RRBs is not so useful later on in the decision process, when more detailed, problem-specific knowledge is needed that is not yet codified in the RRBs. As indicated earlier, this is intentional, because the high-risk decisions are all made early in the design process.

It is more efficient to evaluate all the options to one decision at once than it is to add options after property prediction. In case 1, when we added a new option after the other options had been evaluated, we needed to re-evaluate all the options because in evaluation, all options are compared to each other.

17.5.2. Advantages and disadvantages

Using the RRB process itself seems like nothing more than good design practice. However, recording all the decision rationale so explicitly as we did in our examples may incur high extra costs. In our cases, the use of the RRB process didn’t incur extra costs, because explicit rationale was needed for the decision process anyway. In projects where there is no natural inclination to record rationale, our approach may be too intrusive to work.

Making use of the knowledge inside the RRBs certainly saves effort. Investing time to make this knowledge reusable for other projects in the same company is subject to the same sort of economical considerations as reusing software. For generating, harvesting and validating design knowledge beyond the scope of organizations, we see a role for the academia, and our future research will take part in it.

Acknowledgements. We thank Barbara Paech for her very constructive feedback on our early drafts and later versions, which have improved the quality of this chapter considerably. We thank the anonymous reviewers for their extensive feedback which has led to numerous further improvements.

References

[1] Alexander C, Ishikawa S, Silverstein M, King I, Angel S, Jacobson M, (1977) A Pattern Language: Towns, Buildings, Construction. Oxford University Press

[2] Avison D, Lau F, Myers M, Nielsen PA (1999) Action research. In: Communications of the ACM 42(1):94–97

[3] Basili VR, Caldiera G, Rombach HD (1994) Experience Factory. In: Marciniak JJ (ed) Encyclopedia of Software Engineering. John Wiley & Sons, vol. 1, pp. 528-532

[4] Baskerville RL (1999) Investigating Information Systems with Action Research (Tutorial). In: Communications of AIS 2, Article 19

[5] Bass L, Klein M, Bachmann F (2000) Quality attribute design primitives. Technical report, SEI, CMU/SEI-2000-TN-017

[6] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1996) Pattern-Oriented Software Architecture, volume 1: A System of Patterns. John Wiley & Sons

[7] Dutoit AH, McCall R, Mistrik I, Paech B (2006) Rationale Management in Software Engineering. Heidelberg: Springer-Verlag

[8] Gamma E, Helm R, Johnson R, Vlissides J (1995) Design Patterns. Addison-Wesley Professional

[9] Fowler M, Rice D, Foemmel M, Hieatt E, Mee R, Stafford R (2002) Patterns of Enterprise Application Architecture. Addison-Wesley Professional

[10] Hofmeister C, Nord R, Soni D (1999) Applied Software Architecture. Addison-Wesley Object Technology Series

[11] Hohpe G, Woolf B (2003) Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional

[12] Lee J, Lai K (1996) What is Design Rationale? In: Moran TP, Carroll JM (eds) Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, Mahwah, New Jersey pp 21-52

[13] MacLean A, Young RM, Bellotti VME, Moran TP (1996) Questions, options and criteria: Elements of design space analysis. In: Moran TP, Carroll JM (eds) Design Rationale — Concepts, Techniques, and Use (Computers, Cognition, and Work). Lawrence Erlbaum Associates, Mahwah, pp 53–106

[14] Wieringa RJ (1996) Requirements Engineering: Frameworks for Understanding. John Wiley & Sons

[15] Witte E (1972) Field research on complex-decision-making processes—The phase theorem. In: International Studies of Management and Organization, Vol. 2, pp. 156-182

[16] Zeist B van, Hendriks P, Paulussen R, Trienekens J (1996) Quality of software products — Experiences with a quality model. Kluwer Bedrijfswetenschappen. Book in Dutch. Website contains all relevant information in English. See http://www.serc.nl/quint-book/index.htm.