Documenting and Detecting Errors in Decision-Making using Rationale

Janet Burge
Department of Mathematics and Computer Science
Wesleyan University
Middletown Connecticut, USA
jбурге@wesleyan.edu

Abstract—One way that errors can be introduced into software decisions is when the people designing and implementing the system make poor decisions either because they failed to consider the relevant criteria or because decisions were based on incorrect assumptions. Rationale management systems can be used to support and monitor the decision-making process and can help detect or even avoid some forms of decision-making errors.

Index Terms—Rationale, decision-making.

I. Introduction

Many of the errors that humans introduce during the software development process occur when someone has made a decision that either was inadvisable from the start or where something seemed like a good idea at the time but caused problems later on. Errors are also introduced when a decision is required but not actually made.

So how does this happen? Poor decisions can be made when developers fail to consider relevant solution assessment criteria, prioritize the wrong criteria when making decisions, fail to consider viable (and possibly better) alternatives, and do not leverage information on past success (and failure) with similar problems. Errors can also occur when decisions involve multiple stakeholders. It may not be possible to reach an agreement, stakeholders may use conflicting criteria, and compromises may not turn out to be the correct decision.

Even well considered decisions can result in errors later on if changing system goals or incorrect assumptions cause the choices made to no longer be valid. This can occur when requirements change, assumptions are found to be, or become, invalid, or when new information impacts earlier decisions.

One place where these errors can be captured and potentially identified is in the rationale for the system. While most commonly referred to as design rationale, rationale can capture information throughout the process from decisions made when identifying requirements through to operations and maintenance [1]. Rationale includes the decision problems that must be solved during development, alternative solutions considered, and the criteria used to evaluate alternatives and make a decision.

The remainder of this paper is structured as follows. Section two defines different types of decision errors. Section three describes how rationale notations and systems can be used to document and detect these errors and where they fall short. Section four provides a summary and conclusions.

II. Types of Decision-Making Errors

The ability to make good decisions depends on a number of different factors:

- Understanding the problem to be addressed,
- Identifying alternative solutions to that problem,
- Understanding the criteria used to evaluate solutions,
- Applying the criteria in the decision-making process to make a decision.

The process is iterative—identifying and selecting an alternative can then generate new problems to be solved. Errors can occur at any point in this process.

A. Problem Understanding

The first step in solving a problem is defining it. While some problems are straightforward, others may be more challenging or even wicked [2]. Difficulties in defining problems can occur when multiple stakeholders disagree on what needs to be done (conflicting goals) or when those making the decision do not possess sufficient knowledge of the problem domain. The former issue requires negotiation while the latter requires knowledge engineering.

Errors that can occur here are:

- **Contentious or Conflicting Problem Statements (U1)** - the problem has been described but not all stakeholders agree that it is correct.
- **Vague Problem Statements (U2)** – the problem statement is imprecise and subject to misinterpretation.
- **Incomplete Problem Statements (U3)** – the problem statement is missing critical details.

B. Alternative Identification

Once the problem has been identified, the next step is identifying candidate alternative solutions. If there are similar problems that have already been solved then those solutions may apply to the new problem. If the problem is new then coming up with alternatives may be more difficult. There are many methods for idea generating (brainstorming, etc.) that can be employed to identify alternatives. It may not be necessary (or desirable) to identify all or many alternatives—only that alternatives can be identified that satisfy the solution assessment criteria. Another issue that can arise in complicated problems is conflicts or incompatibility between alternatives.

Errors that can occur here are:

- **Insufficient alternatives (AI1)** – there are alternatives that should be considered as solutions that are missing.
• Infeasible alternatives (AI2) – there are alternatives proposed that can not be implemented.
• Unsupported alternatives (AI3) – there are alternatives selected without sufficient justification.
• Conflicting alternatives (AI4) – there are alternatives selected that conflict with each other.

C. Solution Assessment Criteria Identification

Solution assessment criteria are needed to differentiate alternatives from each other and to ensure that the decisions made meet the system goals. Criteria come in many forms, including functional requirements, non-functional requirements, and assumptions.

Applying criteria to make a decision is further complicated by the possibility that the system goals that provide the criteria may change over time. Current assumptions may not hold true in the future and priorities of the various criteria may change with user needs.

Errors that can occur here are:
• Missing criteria (CI1) – there are important criteria that should be considered when making decisions that have not been examined.
• Changing criteria priorities (CI2) – the importance of criteria changes over time as either more information is discovered or requirements change.
• Changing assumptions (CI3) – assumptions that are initially valid end up changing as the system environment or context changes in unanticipated ways.
• Incorrect assumptions (CI4) – assumptions were made based on incomplete or incorrect information and are later found to be invalid.

D. Applying Criteria in Decision-Making

As with problem identification, issues arise with multiple stakeholders who may prioritize different criteria differently. It also can be challenging to understand how well an alternative does or does not satisfy some criteria. In addition to conflicts from stakeholders, criteria often conflict—requiring tradeoff evaluation when making a decision.

Errors that can occur here are:
• Incorrectly applied criteria (AC1) – decisions are made where the criteria used are not relevant to the alternatives under consideration, such as claiming that a solution is scalable when it is not.
• Failure to consider relevant criteria (AC2) – making a decision on a partial set of criteria where critical criteria are missing.
• Failure to consider tradeoffs (AC3) – critical interplays between criteria are not considered. For example, choosing a more general solution may increase flexibility but make maintenance more challenging.

III. Use of Rationale

One way to capture decision-making and assist with error detection and prevention is by capturing the decision-making process as Rationale. Rationale has been used in many different domains including Human Computer Interaction [3], Engineering Design [4], and Software Engineering [5]. Many rationale approaches use a structured argumentation representation to capture decisions, alternatives, and relationships between them. Some of the earliest notations are Toulmin’s claims, data, warrants, backings, rebuttals, and qualifiers [6]; the Issue Based Information System’s (IBIS) [7] issues, and positions; and Questions, Options, and Criteria’s (QOC) [8] questions, options, and criteria. The IBIS notation was used as the basis of Lee’s Decision Representation Language (DRL) [9]. DRL is one of the most comprehensive notations and extends the basic IBIS notation by adding questions, procedures, groups, and viewpoints.

DRL served as the basis of the RatSpeak notation [10] used in the Software Engineering Using RATionale (SEURAT) Rationale Management System (RMS) [11], an Eclipse plug-in developed to support software maintenance. SEURAT captures decisions (decision problems), alternatives, arguments for and against the alternatives, and questions that need to be answered (which can apply to decisions or alternatives), along with the answers and the procedure followed to get the answers. Arguments come in multiple types—requirements, assumptions, claims (typically non-functional requirements), and dependencies between alternatives. SEURAT also captures background knowledge such as common tradeoffs (claims that occur on opposite sides of an argument) and co-occurrences (claims that occur on the same side of an argument).

SEURAT can inference over the rationale to detect potential problems with either the completeness or consistency of the rationale and the decision-making. Problems detected include:
• Decisions where no alternative had been selected;
• Alternatives that were selected without arguments in their favor;
• Alternatives that were selected with less support than other alternatives also proposed for the same decision;
• Unanswered questions recorded in the rationale;
• Conflicting arguments for and against an alternative;
• Requirement violations;
• Tradeoff and Co-occurrence violations; and
• Conflicts between alternatives where either an alternative was selected that cannot exist with another selected alternative or where one alternative requires another that was not chosen.

A key feature is the ability to use the arguments and their importance (priority) to calculate a numeric score for each alternative. This allows the developer to use SEURAT to perform “what-if” analysis by re-calculating support for alternatives if some of the arguments for and against them changed. This can be done by changing the importance of some claims at a global level or by disabling a requirement or assumption. For example, SEURAT was used to capture the rationale for a conference room scheduling system initially developed as a class project. The most commonly occurring argument was “easy to code.” If this claim is made “not important” many of the existing decisions are flagged as no longer being good choices.
Figure 1 shows an example of some rationale stored in SEURAT for the decision “How to persistently store data in the system?”. The decision has an error overlay icon because neither of the two alternatives has been selected.

- **How to persistently store data in the system?**
  - relational database
  - overkill for the prototype
    - no need to store much data for the prototype
  - can save large numbers of meetings
  - object serialization
  - can easily save collections of objects

Fig. 1. Rationale Example

### A. Rationale to Support Problem Understanding

Problem understanding was a critical motivation behind developing the IBIS system. Horst Rittel’s theory of Wicked Problems proposes a class of problems, in which many design problems fall, called “Wicked Problems.” A key property of these problems is that there is “no definitive formulation.” To address these problems, he proposed an “argumentative process” [2], which was supported by using IBIS.

As mentioned earlier, a key issue in problem understanding and definition is gathering input and achieving consensus with multiple stakeholders. Conklin’s Dialog Mapping process [12] uses Compendium [13][14], an IBIS-based tool, to gain a shared understanding of a problem and its associated issues. Capturing issues and arguments from the team is a key step towards problem understanding and team consensus.

The other challenge is defining, documenting, and addressing knowledge needed to make decisions. DRL and SEURAT are examples of systems that allow the designer or developer to record unanswered questions as part of the rationale. SEURAT will flag these unanswered questions as errors in the rationale until they have been resolved. It also provides a place to record the answer discovered and the procedure followed to obtain the answer, whether it involved asking an expert, reading additional documentation, or running a simulation or experiment.

A better understanding of the problem, including input from all stakeholders, is a step towards avoiding situations where problem definitions are contentious (U1), vague (U2), or incomplete (U3). An argumentative approach with computational support from an RMS can support this goal.

### B. Rationale to Support Alternative Identification

Traditional software documentation records what was done but seldom documents alternatives considered and then rejected. Argumentation-based rationale provides a way to record this information. It also could support the re-use of alternatives generated for similar systems or earlier versions of the same system. This would allow new projects to learn from the success or failure decisions made during prior ones. If an alternative has already been evaluated as infeasible for a similar problem (AI2) this information would be invaluable.

Explicitly recording alternatives and their criteria may reduce the likelihood of selecting unsupported alternatives (AI3). The rationale can be reviewed as part of the development process to ensure that decisions made are made for the correct reasons. Students using rationale in class assignments felt that it encouraged them to consider more alternatives [15](AI1).

A critical feature of SEURAT is the ability to capture the dependencies between alternatives. If two alternatives conflict, the developer should be alerted (AH4). Similarly, if one alternative requires another this should also be enforced.

### C. Rationale to Support Criteria Identification

There different types of reasons for making decisions. At a functional level, there are the software requirements, use cases, or user stories. There are also many forms of non-functional criteria including the “ilities” (scalability, usability, etc.) [16] and other quality-focused goals. There are also assumptions that play a role in the decision-making process, such as assumptions on how the system will be used, the context in which it will run, and other factors that may change over the system’s lifetime.

Rationale provides a place to record these criteria and how they relate to the designed product. As with alternatives, it may be possible to re-use already existing rationale, which can help prevent criteria from being missed (C11).

Ideally a rationale management system should allow criteria to be provided in two ways: as general system criteria that can be selected as decisions are made and as decision or alternative-specific criteria that are only identified when used to differentiate between specific alternatives.

The RMS should also give a global view of criteria being used. For example, SEURAT lets the developer query the rationale to get a list of the different types of arguments used and the types of criteria that they map to. For example, the developer can get a list of all the assumptions that impacted the decision-making process or a list of which arguments (and criteria) appear the most often. Query results gather the criteria being used in one place to support decision-review. These queries can also be used to identify instances where criteria have been proposed as being important to the system yet do not appear in the rationale.

As mentioned earlier, SEURAT will recalculate support for alternatives if criteria priorities change (C2) or assumptions change (C13) or are found to be invalid (C14).

### D. Rationale to Support Criteria Application

Rationale captures the relationships from criteria (requirements, NFRs, assumptions, etc.) to decisions. Using rationale will not prevent a developer from generating incorrect arguments (AC1), but it can be used to provide an easy way to document these arguments so that they can be reviewed. It also can perform some “sanity checks” to look for structural issues and inconsistencies (AC2).
For example, if a common vocabulary is used to identify criteria a rationale management system can detect if the same criteria is used on both sides of an argument. SEURAT provides an “argument ontology” as a base set of claims to allow this comparison to be performed. SEURAT also makes use of tradeoffs and co-occurrences stored as background knowledge to check to see if claims are missing from the arguments (AC3).

E. What Rationale Won’t Do (Yet?)

While rationale is a useful tool in preventing and detecting errors in decision making there are limitations. These systems are only as good as the data captured by them—some forms of incompleteness can be detected, such as the failure to select an alternative or the selection of unsupported alternatives, but many can not be. For some types of errors there is no way to determine if they indicate incorrect reasoning or if they indicate incompleteness in the rationale. The systems can help developers assess the impact of changing criteria but they can not detect when things change.

Another feature not available in existing rationale management system would be the ability to flag elements as having “expiration dates.” This could be a step towards proactively detecting when decisions should be revisited. This could be applied to many different rationale elements but would be particularly useful when attached to assumptions.

Working with multiple stakeholder rationale and using that to do automatic rationale checking is another interesting problem. Scoring can become intractable if rationale can include arguments about arguments but keeping track of who proposed an alternative and who argues for and against could also be useful information. There may be cases where it would be useful to re-evaluate alternatives if someone who has advocated for them or argued against them is no longer involved with the project. Different stakeholders may have different priorities when involved in decision-making.

IV. Summary and Conclusions

Using a rationale management system as a tool in the design and development process supports the detection and in some cases, potentially prevention, of many forms of user error in the decision-making process. For errors that are not detected, such as incorrectly assessing alternatives with respect to the decision criteria, the tool still makes it easier to gather decision information into a form that can be easily reviewed.

Acknowledgment

This work was supported by NSF CAREER Award CCF-0844638. Any opinions, findings, and conclusions or recommendations ex-pressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References


