

A Case Base for Requirements Engineering: Problem Categories and Solution Techniques

Klaas Sikkel, Roel Wieringa, Rolf Engmann

Faculty of Computer Science, University of Twente
PO Box 217, 7500 AE Enschede, Netherlands
{sikkel, roelw, engmann}@cs.utwente.nl

Abstract. We introduce a notion of business problem frames, categorizing the type of IT requirements problems found in organizations, as opposed to Jackson's problem frames which describe a problem in terms of the solution to that problem. A survey of students' projects showed that this a viable notion. We intend to build a case base of business problem frames, and their solutions, as a basis for further research and guidance to practitioners.

1 Introduction

A problem frame is a way of structuring a problem, of partitioning it into principal parts, such that solutions are easier to find than they would be with a different frame. Software Engineering work in patterns [2] and architectures [1, 5] can be viewed as an attempt to define an inventory of problem frames for programmers and software architects. Michael Jackson's problem frames do the same for software requirements [3]. We call these *requirements problem frames*.

Many of our students do a Master's project involving requirements analysis outside the university. We hypothesized that requirements problem frames would be a useful starting point for these student projects, but this appeared not to be true. Jackson's frames feel like *solution* frames: they describe what a good problem decomposition looks like, but provide little help in *how* to decompose a problem. Our students also tend to think in solutions, but with one important difference. How to analyze, frame and occasionally reframe a problem is a skill they have not yet mastered.

In order to guide the analysis process, we investigated whether we could identify *business problem frames*, describing the IT problems that organizations are confronted with, as opposed to the solutions that can be applied to these problems.

By analyzing several dozen students' projects, we identified a small number of often recurring business problem frames. We conjecture that it is a valid notion worth to be pursued. We intend to further consolidate and refine our taxonomy of frames and develop a case base of project reports structured according to these frames. With an additional set of guidelines, supported by evidence from the case base, we hope to train novice requirements engineers to become reflective practitioners, skilled in framing and reframing problems.

2 Problem categories

We started reviewing students' projects without a particular classification scheme, and found that the problem descriptions clustered into a small number of categories.

An enumeration of problem categories

The following problem categories emerged in a bottom-up analysis.

1. *Unidentified problems.* "We know that there is something wrong, but we do not know what is wrong." The real problem has yet to be identified. These are the hardest problems to solve because not even the criteria by which a solution proposal should be evaluated, are known.
2. *COTS type.* "We know what our problem is, and we are looking for a system that can solve our problem. We do not want to build something. Which type of COTS (commercial off the shelf) system would help us?" For example, would a simple email system help us or do we need an advanced groupware system?
3. *COTS product.* "We know what our problem is and we know what kind of functionality we are looking for. Which particular COTS system should we buy?" For example, we want to buy a workflow management system. Which vendor do we choose?
4. *COTS extension.* "We have a COTS system that works satisfactorily, and we want to do other things with the system as well. Can we extend the functionality of our system?" For example, we have an ERP package and we want it to include parts of our business.
5. *New functionality.* "We know what our problem is and now we want to obtain a system that solves it. Which functions should this system have?"
6. *Extended functionality.* "We know what our problem is and our current system does not solve it adequately. How should we change the functionality of this system in order to solve our problem?"
7. *Architecture.* "We know what our problem is and our current system does not solve it adequately. We also know which functionality our system should have. Which revised architecture should this system have in order to solve our problem?"

A systematic classification of problems

Having identified seven problem categories in bottom-up fashion, the question arises whether this is a logical classification, or merely a haphazard partitioning of the search space. A more systematic top-down analysis provided a foundation for these categories. A problem is assigned to a category by answering a small series of questions. A full decision tree is shown in Figure 1.

Q1: Is it known what the problem is?

If not, the task is to identify the problem.

Q2: Is it known what the solution is, i.e., which system functionality could solve the problem?

Requirements analysis typically should identify desired functionality, but some interesting requirements problems may remain when answered affirmatively. We consider both cases.

(i) Problem known, functionality to be determined

We continue with the next two questions.

Q3: Should the current system be continued?

That is, should the current system be extended or replaced?

Q4: Is it a constraint that the new/extended system should be COTS?

A decision to look for COTS constrains the search space and influences the way in which the requirements analysis is to be performed: the emphasis is on evaluating which COTS system can contribute to solving the problem. Otherwise, the emphasis is on analyzing which functionality is needed.

(ii) Problem and desired functionality known

This gives rise to two interesting cases.

Firstly, when a new COTS system is wanted, the appropriate system has to be selected (category 3). This is rather different from category 2, where it is unknown which type of system would solve the problem. Here we know the type of system we are looking for, but a particular vendor has to be selected. It is not uncommon that the functionality of competing products is very similar; features that truly distinguish

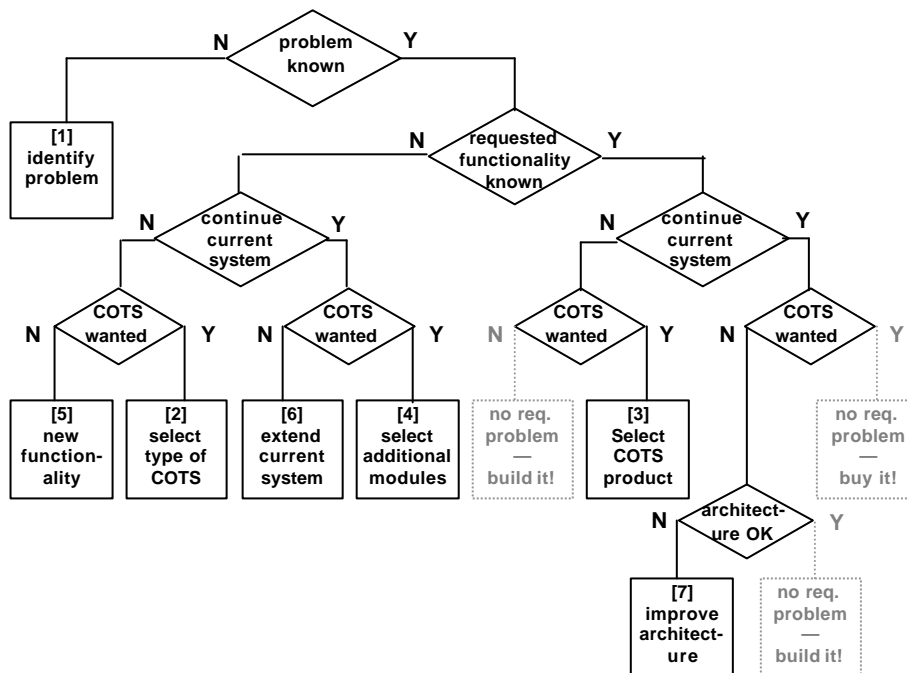


Figure 1: decision tree for business requirements problem classification

these products are more likely to be nonfunctional [4].

Another interesting case arises when the extension of a system implies a nontrivial restructuring. If you know which custom extension you want, the appropriate question is

Q5. Does the current system architecture support the requested extension of the system?

Typical nonfunctional problems are performance bottlenecks and maintainability of a heterogeneous collection of subsystems grown over the years. The task is to design a better system architecture with the same functionality.

Applicability of the classification

Having derived the classification of business problem frames, the question arises how wide the applicability of the framework is. We note the following issues.

Requirements problems outside the scope of this classification

It is assumed that there is a *single* major requirements issue. However, there are situations with multiple, interrelated problems. For example: A major bank wants a new architecture for all their IT while at the same time it is not clear which functionality should be supported by this architecture three years from now. These are wicked problems that are very hard to solve well and this is not the type of problem you would commission to a Master's student—but it has to be acknowledged that, obviously, such projects cannot be addressed by framing it as a single business problem.

Borderline cases

In some cases the answer to a question is not a definite yes or not but e.g. “likely” or “probably”. A typical case is that a COTS solution is the most likely option but not a precondition.

An example: The administration of a city in the Netherlands is establishing a network of branch offices in different parts of the city. A pilot study with an experimental branch office revealed that customer relation management (CRM) is a problem and the software used is highly inappropriate. A new CRM package is needed. The problem is known (Q1: yes) but the functionality of a solution system still to be determined (Q2: no). The current system should be discarded (Q3: no). Is a COTS solution sought? Good question. The IT department of the city could produce a system (they built the current one!), but it is estimated that buying a CRM package would be a lot easier and better.

In this situation, as outlined above, framing the problem as category 2 (COTS) or 5 (custom built, or perhaps COTS) makes a difference for how to address the problem. In the latter case you do a classical requirements analysis, starting with eliciting the particular needs of the users. In the former case you would evaluate the functionality of COTS CRM software at an earlier stage and use that to constrain the solution space. Ex post we can observe that in this particular case frame category 5 was

preferred for pragmatic reasons (a priori exclusion of the city's IT department would have had certain disadvantages) but at the time no explicit choice was made.

Even though it is hard to classify the problem in the given taxonomy, we conjecture that it would have been helpful to have the taxonomy, so as to bring out the issues involved and make explicit which ways there are to address the problem. In the given example it is likely that an earlier study of COTS solutions would have increased the effectiveness of the project.

The surface problem is not the real problem

One should always keep an open eye for the possibility that the problem was framed wrongly and should be reframed. Any approach in requirements engineering should be sensitive to the fact that it is not uncommon that the problem for which a solution is sought is but a symptom of a deeper underlying problem.

3 Structuring Business Problems

We can identify a short list with generic questions that can be asked for each of above problem frames—but the answers are different for different frames.

- (a) *What are the principal parts of the problem?*
- (b) *Which questions can we ask to learn more about the problem?*
- (c) *Which techniques can we use to obtain answers to these questions?*
- (d) *Which solutions patterns—i.e., solutions known to have worked in the past—can be found?*
- (e) *Which techniques can be used to describe those solutions?*

Questions (a-b) are part of the problem frame, in the sense that they address what the problem *is*, whereas questions (c-e) refer to ways of how the problem can be *solved*. Note that Jackson's problem frames relate to question (d).

The guidance that we can give to a (novice) practitioner takes the form of a checklist of questions that could be relevant and techniques and method fragments that could be used. There is no general method that works in all cases; the practitioner remains responsible for his own judgement as to which parts of the problem frame and the solution techniques are applicable in a given business context.

In order to further substantiate this, we intend to build a case base with structured summaries of requirements analysis projects. We could fill the case base with roughly a hundred cases from the past and we expect an influx of several dozen new cases per year. The crucial question is how to structure the case base. We have a first draft of a list of semi-closed questions describing salient aspects of requirements analysis projects, but we envisage several iterations until we have a stable structure.

4 Discussion

Any method for addressing practical problems has to seek a balance between generality and situatedness. Method Engineering (ME) aims to allow practitioners to

configure a method from existing fragments as they see fit within a given context. At this basic level, our approach of business problem frames fits well with ME.

The rise of CASE tools has made it more difficult to combine method fragments; much ME research effort in Software Engineering has gone into the development of appropriate metatools, also known as CAME tools, and empirical research in this area has been somewhat neglected [6]. We consider it very important that our research into business problem frames is grounded on empirical data.

Conclusions

Michael Jackson's problem frames are very useful to organize a problem as a collection of solvable subproblems. However, they provide little help how to decompose a problem.

We have attempted to make a categorization of *business problem frames*, which address an IT requirements problem in terms of what the problem is, not in terms of the proposed solution. First findings show that this notion is useful, as it help clarify which kind of problem is at stake and gives some guidance about how to address the problem. This guidance takes the form of checklist of points to consider and possibly useful techniques. Most problems quite naturally fit into one of the problem categories. For borderline cases it does no harm to consider these from multiple perspectives.

It should be noted that these are indeed first findings, and a starting point, rather than a conclusion, for a type of research that we consider worthwhile to pursue.

To that end we intend to establish a case base of requirements projects. The purpose of this case base is threefold: (1) use the gathered empirical material to further delineate the idea of business problem frames, (2) improve guidelines in the form of checklists on how to address a problem that suits a given frame, and (3) allow the case base to be searched for similar cases by students and others working on a requirements problem.

References

1. Bass, L., Clements, P. and Kazman, R. (1998). *Software Architecture in Practice*. Addison-Wesley, Reading, MA.
2. Gamma, E., Helm R., Johnson, R. and Vlissides J. (1994). *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
3. Jackson, M.A. (1995). *Software Requirements and Specifications: A lexicon of Practice, Principles and Prejudices*. ACM Press, New York.
4. Maiden, N.A. and Ncube, C. (1998). Acquiring COTS Software Selection Requirements. *IEEE Software*, 15(2):46-56.
5. Shaw, M. and Garlan D. (1996). *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, Englewood Cliffs, NJ.
6. Tolvanen, J.P., Rossi, M. and Liu H. (1996). Method Engineering: Current research directions and implications for future research. In S. Brinkkemper, K. Lyytinen and R.J. Welke (Eds.) *Proceedings IFIP TC8 WG 8.1/8.2 Working Conference on Method Engineering*, Chapman & Hall, London.