Deconstructing the Refactoring Process from a Problem-Solving and Decision-Making Perspective

Thorsten Haendler[1] and Josef Frysak[2]
[1]*Institute for Information Systems and New Media, WU Vienna, Austria*
[2]*Institute of Business Informatics - Communications Engineering, JKU Linz, Austria*
thorsten.haendler@wu.ac.at, josef.frysak@jku.at

Keywords: Software refactoring, decision-making process, decision problems, refactoring process model, managing technical debt.

Abstract: Refactoring is the process of improving a software system's internal technical quality by modifying and restructuring a system's source code without changing its external behavior. Manual identification and assessment of refactoring candidates as well as planning and performing the refactoring steps are complex and tedious tasks, for which several tools and techniques for automation and decision support have been proposed in recent years. Despite these advances, refactoring is still a neglected part of software engineering in practice, which is attributed to several barriers that prevent software practitioners from refactoring. In this paper, we present an approach for deconstructing the refactoring process into decision-problems and corresponding decision-making sub-processes. Within this, we pursue the question of whether and how a theoretical perspective can contribute to better understand the difficulties in the refactoring process (*barriers*) and to help improving the refactoring support techniques (*enablers*). For this purpose, we follow a deductive reasoning approach by applying concepts from decision-making research to deconstruct the refactoring process. As a result, we present a process model, which integrates primary decision problems and corresponding decision-making sub-processes in refactoring. Based on this process model, software companies can gain a better understanding of decision-making in the refactoring process. We finally discuss the applied procedure and reflect on limitations and potential of applying such a theoretical perspective.

1 INTRODUCTION

Refactoring is the process of improving a system's internal technical quality by modifying and restructuring a system's source code without changing its external behavior (Fowler, 2009). Manual identification and assessment of refactoring candidates (e.g., bad smells) as well as planning and performing the refactoring steps are complex and tedious tasks, for which several tools and techniques for automation and decision support have been proposed in recent years; such as smell detectors, refactoring-recommendation tools and quality-analyzer tools, see, e.g., (Fontana et al., 2012; Fernandes et al., 2016; Campbell and Papapetrou, 2013).

However, despite these advances, refactoring still seems to be a neglected part of software engineering. A recent survey (Tempero et al., 2017) conducted with 3,785 software developers in software projects using object-oriented concepts shows that practitioners mostly understand the value of refactoring, but are often prevented from doing it. The study identified seven main factors (called *barriers*) that affect the practitioners' decision of whether or not refactor. In particular, these barriers to refactoring are categorized into the following seven categories: missing *resources*, the *risk* to introduce an error, the *difficulty* to perform the refactoring, an unclear *ROI*, *technical* issues, constraints set by the *management*, and a lack of appropriate *tools* support (Tempero et al., 2017).

The identified barriers affect decision makers on different organizational levels: besides decisions on the operational level (e.g., the difficulty to perform the refactoring), there are also decisions located on management level (e.g., ROI); few are on both levels and interrelated (e.g., the allocation of resources). Due to the management issues addressed by the barriers and due to the relevance of technical debt for software projects (Kruchten et al., 2012), we suggest a more general problem-solving and decision-making perspective on software refactoring. In this paper, decision-making is understood as a sub-area

of problem-solving, which requires the selection of a single one of at least two alternatives, that is, mutually exclusive actions. These actions are designed to solve a complex problem by achieving one or more goals. However, an option may be to take no action at all, or to postpone the decision and to search for more information. In the case of refactoring, however, the latter alternative measures are likely to increase the technical debt. On the other side, the resources freed up can then be used in other areas of software development and maintenance. Hence, the dilemma, whether or not to engage in refactoring, and which refactorings to apply in order to maximize the benefits arising from the expenditure of resources, is a key aspect of decision-making in refactoring.

In this paper, we propose a theory-driven approach for deconstructing the refactoring process into decision-making steps and for investigating the character of the included decision problems. Research in decision-making provides multiple concepts that are very promising for re-structuring the refactoring process in order to better understand the included difficulties. As a result, we propose a first version of a process model for decision-making in software refactoring (see Fig. 2) which can help software practitioners in better understanding the dependencies between refactoring activities as well as between the different decision-maker levels. This process model requires further empirical evaluation, which will be approached in the next step. However, we illustrate by example that our process model already can help in understanding refactoring difficulties by allocating refactoring barriers and support techniques to corresponding affected and addressed process steps.

The remainder of this paper is structured as follows. In Section 2, we apply a deductive methodology for deconstructing the refactoring process into process phases and decision problems. In Section 3, we present as a result our process model for decision-making in software refactoring, which reflects on three primary decision problems and corresponding sub-processes. Section 4 illustrates the applicability of the process model by allocating refactoring barriers and support techniques to process steps. In Section 5, we discuss the performed procedure, reflect shortly on related research and describe limitations and potential of this approach. Section 6 concludes the paper.

2 DECONSTRUCTING REFACTORING

In this paper, we aim at deconstructing the decision-making process for refactoring. By deconstruction, we understand the identification of individual steps/tasks, characteristics of decision problems, decision-makers involved as well as the dependencies between the steps in the refactoring process from the perspective of problem solving and decision-making. A result of this procedure is an integrated process model which reflects on these aspects and aims at helping better understand decision-making in refactoring, see Fig. 2. For this reason, we apply deductive reasoning, also known as *top-down logic*. The reasoning starts with stating two or more (often theory-based) linked premises which lead to a new hypothesis or conclusion (syllogism), see, e.g., (Evans et al., 1993). In our case, we investigate whether deducing concepts and classification schemes from general decision-making research to the refactoring process can help better understanding the difficulties in refactoring. Accordingly, the following syllogism was applied:

- ***major premise***: *decision-making processes can be structured into certain phases/steps*, see, e.g., (Simon, 1977).
- ***minor premise***: *refactoring is a decision-making process including multiple decision problems*, see, e.g., (Leppänen et al., 2015).

Following on these premises, the ***hypothesis*** can be concluded that *the refactoring process can be structured into the phases or steps of a decision-making process for solving decision problems*. In a broader sense, this hypothesis also raises the questions *to which extent refactoring can be understood as a decision-making process and which decision-making problems are included*? In order to verify this hypothesis and answer these questions, we will describe in the following the main concepts on process models and decision problems from refactoring on the one hand and from a decision-making perspective on the other.

2.1 PROCESS MODELS

Process models for refactoring. For refactoring only very few process models are established. Since refactoring can be understood as a specific maintenance activity aiming at improving maintainability of the software system (in terms of a preventive maintenance), we also included process models for software maintenance, of which several are established, for instance, (Boehm et al., 1981; Osborne, 1987; Kitchenham et al., 1999). Throughout all these variants, the process is driven by the occurrence of a problem (e.g. triggered by a change request or a problem report). Very similarly most models consist of the three key phases *comprehension* (or investigation), (code) *mod-*

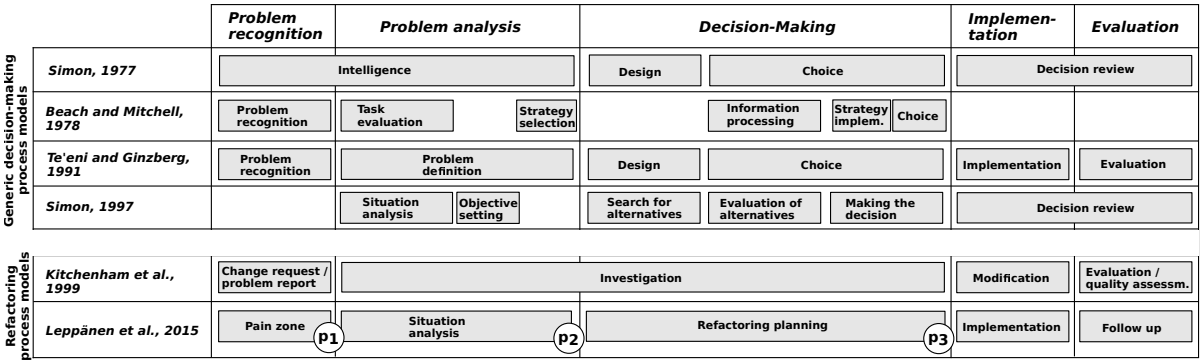

Figure 1: Mapping decision-making process models (top) (Simon, 1977; Beach and Mitchell, 1978; Te'eni and Ginzberg, 1991; Simon, 1997) to refactoring process models (bottom) (Kitchenham et al., 1999; Leppänen et al., 2015).

ification and *evaluation*. In all mentioned models only very little focus is set on decision-problems and corresponding steps of decision-making. In particular, the model in (Kitchenham et al., 1999) describes that after the investigation phase, the maintenance manager must decide whether or not to perform the modification. A recent example for a process model explicitly for software refactoring is given in (Leppänen et al., 2015). The model represents the result of a case study with software developers from three Finnish software companies on their daily refactoring practices. The resulting decision-making process framework consists of the key phases *pain zone* (e.g., triggered by a technical debt, poor design or new requirements), *situation analysis, refactoring planning* (including discussions on requirements and possible solutions as well as task allocation), *implementation* (i.e. the actual code modification), and finally *follow up*, which represents the evaluation of the refactoring in the broadest sense. The process steps of Kitchenham et al. and Leppnen et al. are depicted in Fig. 1 (bottom) and are contrasted by process models for decision-making (top in Fig. 1; see below).

Decision-making process models in general. The probably most well known approach to separate decision-making processes into single phases from the early days of decision-making research is the four phases approach of Simon (Simon, 1977; Pomerol and Adam, 2004), which in particular comprises the phases *intelligence, design, choice* and *review*. During the *intelligence* phase, the environment is observed to detect cues indicating problems, which potentially initiate a decision. Once such cues are identified, the underlying problem is then analyzed in more detail by collecting additional information on that problem. During *design* phase, alternative solutions for the previously identified problem are derived. At this phase, gathering further information is an integral part as well. Now, however, information search aims at supporting the generation of alternative solutions to the previously identified problem. In the *choice* phase, the alternatives are then evaluated according to various criteria. The end of this phase is usually marked by the selection of one of the alternatives. With the solution selected, deemed most adequate to deal with the detected decision problem, the solution is finally implemented and the outcomes observed. *Reviewing* the observed outcomes in relation to the ones estimated during decision-making then allows to determine the effectiveness and accuracy of the decision-making strategy for the particular case.

Still high in popularity in research community, this model was expanded by many authors, see, e.g., (Asemi et al., 2011; Te'eni and Ginzberg, 1991; Courtney, 2001; Huber, 1980; Mora et al., 2005), who added additional phases or broke down the process into more detail phases. Te'eni and Ginzberg (Te'eni and Ginzberg, 1991), for instance, while agreeing on single design and choice phases, split the intelligence phase into a problem recognition and problem definition phase, and the review phase into a phase for implementing the solution and one for evaluating the outcomes. In addition, in their model they suggest that this process is repeated iteratively.

Later, Simon also published an extended version of his phase framework, see (Asemi et al., 2011). Within this framework, the review phase is still positioned at the end of the process and the search for alternatives phase corresponds to the design phase in the small version. The intelligence phase, however, has been replaced by a phase for situation analysis and a subsequent phase, in which the decision goals and criteria are in the focus. Furthermore, instead of where the choice phase was situated, the process is divided into a phase for evaluating alternatives and the final decision-making phase.

From a slightly different perspective, in their ap-

proach, Beach and Mitchell (Beach and Mitchell, 1978) describe "a typical model of individual decision-making" to explore how a single decision maker adapts his cognitive information processing to different decision situations. This model was added, as it shows how people identify a problem, evaluate the problem, and then choose an appropriate decision-making strategy based on that assessment. Once a decision strategy deemed suitable for the decision problem has been selected, the information processing including the search for and evaluation of information is started in a further step. This step is then followed by the implementation of the chosen decision strategy and finally the selection of the best solution.

The process of Beach and Mitchel (top) Te'eni and Ginzberg (second), and the latest version by Simon (third) are presented in parallel in Fig. 1.

Findings. At the first glance, the mapping between of the generic decision-making models and the process models for refactoring provides some obvious similarities, since, at a high level, the refactoring process (e.g., the one by (Leppänen et al., 2015)) can be fitted into the five phases of which the first two are problem-oriented: *problem recognition* (pain zone) and *problem analysis* (situation analysis); the third and fourth are decision-oriented: the actual *decision-making* (refactoring planning) and *decision implementation* (performing the refactoring, i.e. the actual code implementation); and finally *evaluation* (in terms of an optional follow up). On a second view, we also identify some differences. Comparing the amount of steps described by generic decision-making approaches to the ones of refactoring, one can see that especially the phases of problem analysis and decision-making are more sophisticated in the area of generic decision-making. Moreover, the generic approaches presuppose that only one major decision problem is in center of the decision-making process. For the refactoring process, this would conclude that refactoring planning addresses the main refactoring decision problem, i.e. for example the identification, comparison and selection of refactoring paths. This obviously contradicts to the barriers observed in (Tempero et al., 2017) (see above). The multitude of barriers and the different organizational levels involved in the process suggest that there are several decision problems included in the refactoring process.[1]

[1]In Section 3, we propose a process model for selected decision-making aspects in refactoring which also includes the findings of 2.2.

2.2 DECISION PROBLEMS

Table 1: Dimensions for characterizing decision problems; dimension (1-9) are adapted from (Grünig and Kühn, 2013), dimension (10) from (Gorry and Morton, 1989).

Dimension	***Characteristics***		
(1) Complexity	Simple	Complex	
(2) Structuredness	Well-structured	Ill-structured	
(3) Solution space	Choice problem	Design problem	
(4) Framing	Threat problem	Opportunity problem	
(5) Interrelatedness	Independent decision problem	Decision problem in a decision sequence	
(6) Problem level	Original decision problem	Meta-problem (Sub-problem)	
(7) Actor type	Individual	Collective (Group)	
(8) Goals/Criteria	Single	Multiple	
(9) Certainty levels	Decisions under certainty	Decisions under risk	Decisions under uncertainty
(10) Hierarchy level	Operational (control)	Management (control)	Strategy (planning)

Decision problems in refactoring. Independent from the process models discussed in the previous section, multiple decision problems (and sub-problems) in refactoring are addressed by research literature, see, e.g., (Fernández-Sánchez et al., 2015; Ribeiro et al., 2016) and Section 4. For the purpose of this paper, we focus on the following three problem areas, which can be located in the process-model mapping in Fig. 2.1 and should be answered after the corresponding phase:

1. *Management of Technical Debt (TD)* (at management level) with the question of ***Whether (and when) to refactor?***, see (p1) in Fig. 1 and, e.g., (Kruchten et al., 2012).
2. *Detection and assessment of refactoring candidates* with the question of ***What to refactor (first)?***, see (p2) and, e.g., (Fowler, 2009; Ribeiro et al., 2016).
3. *Refactoring planning and performing the actual refactoring steps* with the question of ***How to refactor?***, see (p3) and, e.g., (Fowler, 2009; Suryanarayana et al., 2014).

Decision problems in general. Decision-making demands for choosing a single out of at least two alternative actions. The action is necessary to solve the problem of achieving one or more objectives. In some cases, like refactoring, it may also be an option to take no action at all or to postpone fixing the decision and continue searching for more information. To classify decision-making problems, a variety of characteristics have been identified. In (Grünig and Kühn, 2013), nine important dimensions of characteristics are distinguished, which are summarized in Table 1 (1-9):

First, a decision-making problem may be perceived either easy or complex, depending on various factors such as the number of information cues or the

Table 2: Phases of decision-making with relevant process input and output as well as characteristics for three selected key decision problems in refactoring.

Dimension	***Whether (and when) to refactor?***	***What to refactor (first)?***	***How to refactor?***
Problem recognition	How to identify/measure technical debt?	How to review the system?	How to identify options?
Problem analysis	How to analyze the problem context?	How to assess the refactoring candidates?	How to compare options?
Decision-making	How to plan resource?	How to prioritize refactoring candidates?	How to plan refactoring steps?
Decision implementation	How to allocate resources?	How to select a candidate?	How to perform the modification?
Evaluation	How to measure ROI?	How to evaluate the effects of refactoring regarding software design?	How to evaluate the effects of the refactoring regarding software behavior?
Process input	Software project	Resources (time frame, tools, developers etc.)	Resources (time frame, tools, developers etc.), refactoring candidates
Process output	Allocated resources (down)	List of candidates (down), Refactored system with intended software design (up)	Refactored system with correct behavior (up)
Information needs	Budget, release plan, project context and state (condensed information)	Rules for candidate identification, prioritization paradigm (detailed information)	Rules for performing refactorings (detailed information)
(1) Complexity	Complex		
(2) Structuredness	Partly ill-structured, partly well-structured (programmable, see Section 4.2)		
(3) Solution space	Choice (by having the option to allocate resources or not) Design (by deciding to which extent resources shall be allocated)	Design (regarding the identification/assessment of candidates)	Choice (by having multiple distinct refactoring options)
(4) Framing	Probably mainly perceived as a threat problem, see, e.g., (Tempero et al., 2017).		
(5) Interrelatedness	to *what to refactor?* (succeeding)	to *whether to refactor (preceding)?* (preceding) and *how to refactor (succeeding)?*	to *what to refactor?* (preceding)
(6) Associated sub-problems	Exemplary sub-problems for each decision problem are stated above as phases of decision-making.		
(7) Actor type	(Project) manager(s)	Software architect(s) and developer(s) (in general: the software design expert)	Software developer(s)
(8) Goals/criteria	Multicriteria		
(9) Certainty level	Decision under uncertainty (Decision under risk, if probability of increase of technical debt is measurable or can be estimated)	Decision under uncertainty (Decision under risk, if probability of introducing new design flaws can be measured or estimated)	Decision under uncertainty (Decision under risk, if chances of refactoring success can be estimated)
(10) Hierarchy level	Management	Operational	Operational

familiarity of the decision-maker with the particular task (Liu and Li, 2012).

Second, a decision-making problem can be either well- or ill-structured. Ill-structured decision-making problems are usually novel problems for which no predefined methods are known to handle them and which often possesses no single, correct solution.

Third, the solution space may either consist of a set of immutable, predefined alternatives to choose from, or may require the decision maker to design the alternatives within a continuous solution space (Yoon and Hwang, 1995).

Fourth, depending on the situation and the view point adopted by the decision maker, a decision may either be recognized as a threat of loss or an opportunity to gain an advantage.

Fifth, a decision-making problem can appear largely independent of other decision-making problems (static), or, as part of a sequence of decisions, can have dependencies on other decisions, as described for dynamic decisions (Atkins et al., 2002).

Sixth, the decision-making problem may represent the originating problem, or may represent a meta-problem, such as how much information to collect for a decision (Grünig and Kühn, 2013).

Seventh, decision-making also heavily depends on the number of stakeholders involved. While some decision-making problems allow decisions to be made autocratically by one person, in other decision-making problems multiple people contribute to the decision-making process, and can thereby influence the decision taken.

Eight, in some decision-making problems, the decision maker only needs to focus on a single objective or goal. However, in practice, a decision-making problem often requires to pursue multiple objectives or goals, which means that several criteria must be considered at the same time in the decision-making process (Yoon and Hwang, 1995).

Last but not least, decision-making problems can be distinguished depending to the predictiveness of their outcomes. In decisions under certainty, the outcomes are known and believed to be certain. For decisions under risk, at least the probability of their occurrence is known. In uncertainty decisions, on the other hand, there are no indications as to whether these will occur as predicted.

Moreover, from a business perspective, decision-making problems can also be classified by the level of management activities. According to the framework of (Gorry and Morton, 1989), three levels can be distinguished in this context: *operational control, management control* and *strategic planning*, see (10) in Tab. 1.

Findings. Based on the characteristics of decision problems in general (stated in Table 1) and the procedural aspects from Sect. 2.1, we characterize in the following the three selected problem areas in refactoring. Table 2 summarizes the findings for the examined decision problems regarding the identified sub-process phases, sub-process in- and output as well as the problem characteristics.

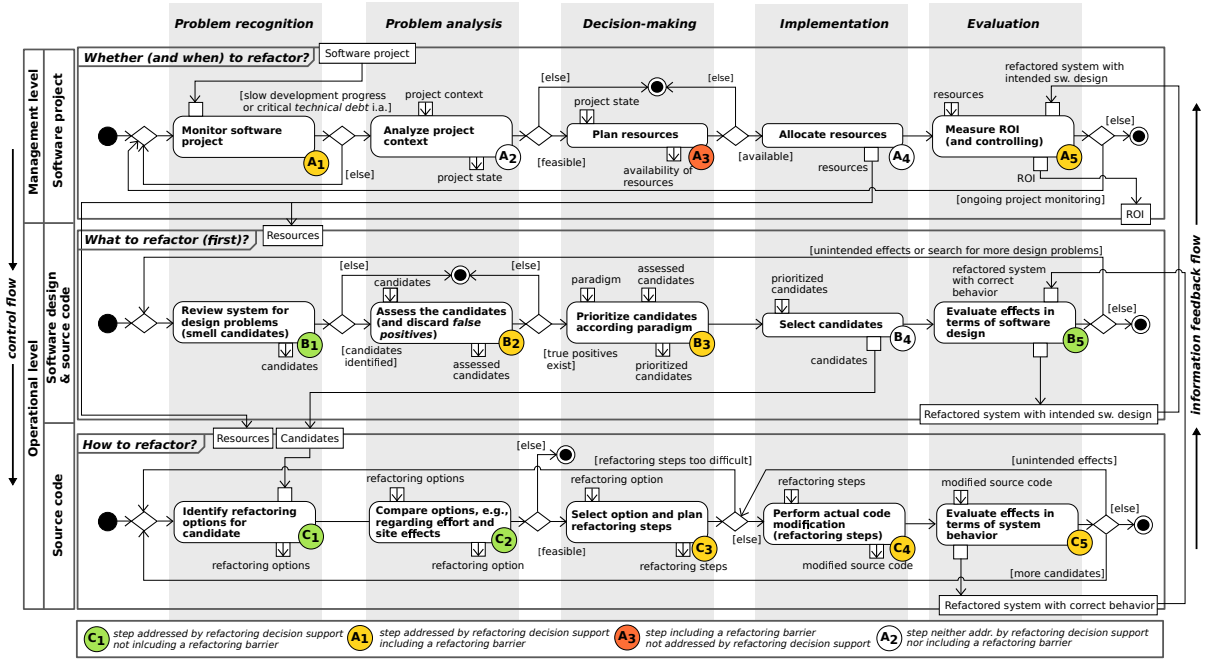

Figure 2: Proposed refactoring process model as result of the applied analysis. The model integrates the decision-making sub-processes for primary decision problems in refactoring: *whether (and when) to refactor?* (top), *what to refactor (first)?* (center), and *how to refactor?* (bottom). Each sub-process is ordered by key phases (horizontal) and by the organizational level of decision makers and context (vertical), for details on the model, see Section 3; for details on the allocation of barriers and support techniques to process steps, see Section 4.

3 A PROCESS MODEL FOR DECISION-MAKING IN REFACTORING

Fig. 2 depicts the proposed process model resulting from deconstructing the refactoring process and from analyzing the characteristics of the selected decision problems (see Table 2). It represents an integrated perspective on three interrelated sub-processes, each focusing on one decision problem. Each decision-making process is structured by the five key phases of *problem recognition, problem analysis, decision-making, (decision) implementation*, and *(decision) evaluation* (as identified in Section 2.1) and specified in terms of a UML2 activity diagram (Object Management Group, 2015).

Organizational levels and communication flows The sub-processes are located on different organizational levels (i.e., management, operational) and focus on different aspects of the software project (i.e., project management, software design/architecture, and source code; as depicted left-hand of the sub-processes). The interrelations between the levels are expressed via different kinds of flows:

- The *control flow* from management to organizational level is represented by inputs for the lower levels in terms of sources (e.g., time frame, people, tools; top-down) and (refactoring) candidates.
- Vice versa, the *information feedback flow* is represented by reporting the results of the evaluation to the specific higher level (bottom-up).

This communication between the three levels corresponds to the (aggregated) information needs of each level described by (Gorry and Morton, 1989).

Refactoring starting points Indicated by the three start nodes in Fig. 2, the process can be started at each of the three levels, be it on the Management level or at both Operational levels.

- Following a *top-down* approach, the process may be triggered by a management decision e.g., driven by noticing a slow development progress (*whether to refactor?*).
- Through allocation of corresponding resources, the manager triggers the underlying sub-process(es). The second sub-process, located at the center of Fig. 2, may also be started directly by a software architect or a developer who detects design flaws while reviewing the system's software design (inside-out, *what to refactor (first)?*; provided that the necessary resources have been formerly allocated).

- Also the bottom sub-process *how to refactor?* can be started based on a previously collected list of candidates. This way a software developer who is actually developing can process a list of refactoring candidates that has been collected before, e.g., via an issue tracker such as *Jira*.

This flexibility in triggering the refactoring process corresponds to (Fowler, 2009) who states that causes and starting points for refactoring source code can be very diverse. Moreover, all sub-processes are cyclic and can repeat, e.g. for multiple candidates or in case the evaluations indicate an error. From a practical perspective, these steps may not all be performed consciously. For example, they may be skipped intentionally or may be done intuitively, see, e.g., (Kahneman, 2011).

An overview of existing support techniques for software developers to address the process steps is i.a. given in the next section.

4 BARRIERS AND ENABLERS IN THE PROCESS MODEL

To illustrate the applicability of our process model in Fig. 2 for supporting software practitioners in understanding the refactoring process, we allocate refactoring barriers and refactoring support techniques to corresponding affected or addressed process steps. In the following the barriers and support techniques are described in more detail.

4.1 REFACTORING BARRIERS

(Tempero et al., 2017) identified seven categories of refactoring barriers which can be allocated to process steps as follows.

- missing ***resources*** (such as the time frame, group size, or tools and technologies); as result of step (A3)) in Fig. 2 with effects on all steps in both underlying sub-processes.
- the ***risk*** of introducing an error; relevant at multiple levels, especially in step (C4) in Fig. 2 and for evaluating whether an error has been introduced (see step (C5)).
- the ***difficulty*** to perform the refactoring which is relevant for step (C4)in Fig. 2 as well as the preceding steps which focus on identifying (step (C1)), comparing (step (C2)), and selecting refactoring option (step (C3)).
- unclear ***ROI***, on management levels in steps (A1) and (A5) in Fig. 2, but also on operational levels, e.g. step (B3).
- ***technical*** issues, e.g., as a lack of technologies or tools (relevant in many steps, e.g., step (C5) in Fig. 2).
- constraints set by the ***management*** which are again result of steps (A3) and (A4) via attributed resources; but also in terms of corresponding control and information flows.
- lack of appropriate ***tools*** (in multiple steps, see below, can also be partially seen as a result of step (A3)).

This allocation shows that some barriers cross the steps and are localized on multiple organizational levels which makes them even harder to handle.

4.2 REFACTORING SUPPORT

In recent years, several techniques and tools have been proposed for decision-support in refactoring. For an overview, see, e.g., (Simmonds and Mens, 2002; Mens and Tourwé, 2004; Mealy and Strooper, 2006; Fontana et al., 2012; Fontana et al., 2015; Fernandes et al., 2016). For the purposes of this paper, the tools and techniques are roughly divided into the following categories. For each group, the addressed process steps in Fig. 2 are stated.

- ***Smell-detection & refactoring recommendation tools*** (such as *JDeodorant* (Tsantalis, 2017) or *Decor* (Ptidej, 2017)) support in (semi-) automatically identifying smell and refactoring candidates via symptoms by analyzing the source code. For this purpose, rules are used which apply metrics and thresholds. Smell detectors address step (B1) in Fig. 2, refactoring recommendation tools also (C1), (C2) and (C3)).
- ***Code-Quality and Design-Critique Tools*** (such as *JArchitect* (CoderGears, 2017) or *NDepend* (ZEN PROGRAM, 2017)) assist software engineers in reviewing the source code or in investigating a system's design and architecture. Most of them provide several visualization techniques (e.g. matrices, graphs) for reflecting static dependencies between system units (e.g., for assessing the *as-is* software design, addressing step (B5)).
- ***Refactoring Tools*** (such as IDEs like *RCP Eclipse* or (Roberts et al., 1997)) provide the automatic or guided/interactive execution of refactoring steps (addressing step (C4) in Fig. 2).
- ***Technical Debt Management and Analysis Tools*** (such as *SonarQube* (Campbell and Papapetrou, 2013), *Sonargraph* (hello2morrow, 2017)) measure and quantify a system's technical debt in terms of a concrete score, mostly in terms of person-hours necessary to fix the debt. For this, they apply metrics and thresholds based on static

analysis techniques. (addressing steps (A1) and (A5), and partially (A3) in Fig. 2 (regarding the estimation of person-hours needed).

- ***Automated Regression Testing Frameworks*** (such as *XJunit* test frameworks) help to ensure that the system still behaves as intended, i.e. that no errors have been introduced by the code modifications (addressing step (C5)).
- ***Documented Knowledge on Refactoring Rules*** Multiple catalogs exist which document rules for performing refactoring-related tasks, such as for symptom-based candidate identification or for performing refactoring steps. Some of this decision-making knowledge has already been implemented into corresponding support systems (see above). In particular, there is documented knowledge for instance available for:
  - *detecting smell candidates* via symptoms (see e.g. (Fowler, 2009; Suryanarayana et al., 2014), addressing step (B1) in Fig. 2),
  - identifying *smell false positives*, see, e.g., (Fontana et al., 2016) (addressing (B2)),
  - paradigms for *prioritizing candidates*, see, e.g., (Ribeiro et al., 2016) (addressing step (B3)),
  - comparing and performing *refactoring steps* (also see e.g. (Fowler, 2009; Suryanarayana et al., 2014), addressing step (C1)).

Some tools also combine certain functionalities, such as for instance *SonarGraph* (hello2morrow, 2017), *JArchitect* (CoderGears, 2017) or *NDepend* (ZEN PROGRAM, 2017).

Table 3 shows the process steps of Fig. 2 with exemplary barriers (included in step) and support techniques (addressing the step). This confrontation illustrates on the one hand that multiple steps are addressed by support techniques which have not been identified as barriers (e.g., (B5), (C1)). On the other hand it becomes evident that some steps are not covered sufficiently by corresponding refactoring support techniques (see, e.g., (A3)).

Table 3: Exemplary allocation of barriers identified by (Tempero et al., 2017) and of several refactoring decision-support techniques to steps in the process model in Fig.2.

No	*Step*	*Barriers*	*Support*
A1	Monitor software project	unclear ROI	TD management and analysis tools
A2	Analyze project context	–	–
A3	Plan resources	missing resources, management dependencies	(TD management and analysis tools)
A4	Allocate resources	–	–
A5	Measure ROI	unclear ROI	TD management and analysis tools
B1	Review system	–	smell-detection tools, documented knowledge on smell detection
B2	Assess candidates	*lack of tool support*	documented knowledge on smell false positives
B3	Prioritize candidates	*lack of tool support*	documented knowledge on prioritization paradigms
B4	Select candidates	–	–
B5	Evaluate effects in (software design)	–	design-critique tools
C1	Identify refactoring options for candidate	–	refactoring recommendation tools
C2	Compare options	–	refactoring recommendation tools
C3	Select option and plan refactoring steps	difficulty of refactoring, *lack of tool support*	refactoring recommendation tools
C4	Perform code modification	difficulty of refactoring, risk of introducing unintended effects, *lack of tool support*	refactoring tools
C5	Evaluate effects (system behavior)	risk of introducing unintended effects	automated regression testing frameworks

5 DISCUSSION

Motivated by the aim to better understand the difficulties in the refactoring process, we applied in this paper a theoretical perspective on decision problems in the refactoring process. For this reason, we used concepts of decision-making for deconstructing the refactoring process. The result of this analysis is a process model for decision-making in software refactoring (see Fig. 2) which comprises the sub-processes of the three interrelated decision problems expressed by the questions *whether, what,* and *how to refactor?* Due to many sub- and meta-problems which are part of every cognitive process, not all probable decision problems in refactoring could have been covered. So, for each sub- or meta-problem, probably a separate decision-making process could be specified.

So far only very few process models for refactoring are available. To the best of our knowledge, the framework proposed by (Leppänen et al., 2015) represents the only process model for refactoring that explicitly includes decision problems. It is based on an empirical study with three software companies (and expressed in terms of a state chart, see Section 2.1). Our approach complements the state-of-the-art of refactoring research by providing a theory-based process description in terms of a process model that integrates three key decision problems and corresponding decision-making processes for refactoring. According to applied concepts of decision-making research, the activities are structured into certain phases of decision-making and along hierarchical levels with corresponding communication flows. Thus, the process model requires an empirical evaluation which will be approached in the next step.

For Section 4, only barriers were presented that have been identified by (Tempero et al., 2017). In

addition, we here only included tools and techniques from the software engineering domain. Probably more related tools and techniques from other domains are available such as from project management, resources management, management information systems or decision support systems.

6 CONCLUSION

In this paper, we applied a deductive approach for deconstructing the refactoring process into distinguished phases, decision problems and corresponding decision-making sub-processes. As a result, we have developed a process model including decision-making steps for three selected major decision-problems in the refactoring process as well as reflecting the characteristics of the decision-making sub-processes on different organizational levels.

We have also shown by example that our model allows for allocating refactoring *enablers* (i.e. refactoring techniques and tools) and *barriers* to process steps, which may help software practitioners in understanding the difficulties in the refactoring process and the relationship between enablers and barriers.

For future work, we plan a survey with software developers and managers to evaluate and refine the proposed process model. Within this, we also seek to investigate the role of support techniques and barriers for each step. Furthermore, we intend to investigate how the decision-support techniques in refactoring and other related domains, such as project and resource management, can be combined, especially in order to support information and control flows between different organizational levels (also in terms of an integrating project cockpit).

REFERENCES

Asemi, A., Safari, A., and Asemi Zavareh, A. (2011). The Role of Management Information System (MIS) and Decision Support System (DSS) for Managers Decision Making Process. *International Journal of Business and Management*, 6(7):164.

Atkins, P. W. B., Wood, R. E., and Rutgers, P. J. (2002). The Effects of Feedback Format on Dynamic Decision Making. *Organizational Behavior and Human Decision Processes*, 88(2):587–604.

Beach, L. R. and Mitchell, T. R. (1978). A Contingency Model for the Selection of Decision Strategies. *The Academy of Management Review*, 3(3):439–449.

Boehm, B. W. et al. (1981). *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ).

Campbell, G. and Papapetrou, P. P. (2013). *SonarQube in action*. Manning Publications Co.

CoderGears (2017). JArchitect. [last access: June 8, 2018].

Courtney, J. F. (2001). Decision making and knowledge management in inquiring organizations: toward a new decision-making paradigm for DSS. *Decision Support Systems*, 31(1):17–38.

Evans, J. S. B., Newstead, S. E., and Byrne, R. M. (1993). *Human reasoning: The psychology of deduction*. Psychology Press.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., and Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, page 18. ACM.

Fernández-Sánchez, C., Garbajosa, J., and Yagüe, A. (2015). A framework to aid in decision making for technical debt management. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pages 69–76. IEEE.

Fontana, F. A., Braione, P., and Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *J. Object Technology*, 11(2):5–1.

Fontana, F. A., Dietrich, J., Walter, B., Yamashita, A., and Zanoni, M. (2016). Antipattern and code smell false positives: Preliminary conceptualization and classification. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 609–613. IEEE.

Fontana, F. A., Mangiacavalli, M., Pochiero, D., and Zanoni, M. (2015). On experimenting refactoring tools to remove code smells. In *Scientific Workshop Proceedings of the XP2015*, page 7. ACM.

Fowler, M. (2009). *Refactoring: improving the design of existing code*. Pearson Education India.

Gorry, G. A. and Morton, M. S. (1989). A framework for management information systems. *Sloan Management Review*, 30(3):49–61.

Grünig, R. and Kühn, R. (2013). *Successful Decision-Making*. Springer Berlin Heidelberg, Berlin, Heidelberg.

hello2morrow (2017). Sonargraph. [last access: June 8, 2018].

Huber, G. P. (1980). Organizational science contributions to the design of decision support systems. *Decision support systems: issues and challenges*, pages 237–275.

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

Kitchenham, B. A., Travassos, G. H., Von Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., Takada, S., Vehvilainen, R., and Yang, H. (1999). Towards an ontology of software maintenance. *Journal of Software Maintenance*, 11(6):365–389.

Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.

Leppänen, M., Lahtinen, S., Kuusinen, K., Mäkinen, S., Männistö, T., Itkonen, J., Yli-Huumo, J., and Lehtonen, T. (2015). Decision-making framework for refactoring. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pages 61–68. IEEE.

Liu, P. and Li, Z. (2012). Task complexity: A review and conceptualization framework. *International Journal of Industrial Ergonomics*, 42(6):553–568.

Mealy, E. and Strooper, P. (2006). Evaluating software refactoring tool support. In *Software Engineering Conference, 2006. Australian*, pages 10–pp. IEEE.

Mens, T. and Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on software engineering*, 30(2):126–139.

Mora, M., Forgionne, G., Cervantes, F., Garrido, L., Gupta, J. N., and Gelman, O. (2005). Toward a Comprehensive Framework for the Design and Evaluation of Intelligent Decision-making Support Systems (i-DMSS). *Journal of Decision Systems*, 14(3):321–344.

Object Management Group (2015). Unified Modeling Language (UML), Superstructure, Version 2.5.0. [last access: June 8, 2018].

Osborne, W. M. (1987). Building and sustaining software maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 13–23.

Pomerol, J.-C. and Adam, F. (2004). Practical decision making–from the legacy of herbert simon to decision support systems. In *Actes de la Conférence Internationale IFIP TC8/WG8*, volume 3.

Ptidej (2017). DECOR. [last access: June 8, 2018].

Ribeiro, L. F., de Freitas Farias, M. A., Mendonça, M. G., and Spínola, R. O. (2016). Decision criteria for the payment of technical debt in software projects: A systematic mapping study. In *ICEIS (1)*, pages 572–579.

Roberts, D., Brant, J., and Johnson, R. (1997). A refactoring tool for smalltalk. *Urbana*, 51:61801.

Simmonds, J. and Mens, T. (2002). A comparison of software refactoring tools. *Programming Technology Lab*.

Simon, H. (1997). The poliheuristic theory of foreign policy decision-making. *Decision making on war and peace: The cognitive-rational debate*, 1:81.

Simon, H. A. (1977). *The New Science of Management Decision*. Prentice Hall PTR, Englewood Cliffs, NJ.

Suryanarayana, G., Samarthyam, G., and Sharma, T. (2014). *Refactoring for software design smells: Managing technical debt*. Morgan Kaufmann.

Te'eni, D. and Ginzberg, M. J. (1991). Human-computer decision systems: The multiple roles of dss. *European journal of operational research*, 50(2):127–139.

Tempero, E., Gorschek, T., and Angelis, L. (2017). Barriers to refactoring. *Communications of the ACM*, 60(10):54–61.

Tsantalis, N. (2017). JDeodorant. [last access: June 8, 2018].

Yoon, K. P. and Hwang, C.-L. (1995). *Multiple Attribute Decision Making: An Introduction*. Sage Publications, Incorporated, Thousand Oaks, CA, new. edition.

ZEN PROGRAM (2017). NDepend. [last access: June 8, 2018].