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

Thorsten Haendler\textsuperscript{1} and Josef Frysak\textsuperscript{2}

\textsuperscript{1}Institute for Information Systems and New Media, WU Vienna, Austria
\textsuperscript{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 Papa-petrou, 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 REFAC T ORING

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-
**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). 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 detailed 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 Mitchell (top) Te’eni and Ginzenberg (second), and the latest version by Simon (third) are presented in parallel in Fig. 1.

Findings. At the first glance, the mapping between 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.\footnote{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

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 \footnote{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 \footnote{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 \footnote{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 (Grüning and Kühn, 2013), nine important dimensions of characteristics are distinguished, which are summarized in Table 1 (1-9):

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Complexity</td>
<td>Simple</td>
</tr>
<tr>
<td>(2) Structuredness</td>
<td>Well-structured</td>
</tr>
<tr>
<td>(3) Solution space</td>
<td>Choice problem</td>
</tr>
<tr>
<td>(4) Planning</td>
<td>Threat problem</td>
</tr>
<tr>
<td>(5) Interrelatedness</td>
<td>Independent decision problem</td>
</tr>
<tr>
<td>(6) Problem level</td>
<td>Original decision problem</td>
</tr>
<tr>
<td>(7) Actor type</td>
<td>Individual</td>
</tr>
<tr>
<td>(8) Goals/Criteria</td>
<td>Single</td>
</tr>
<tr>
<td>(9) Certainty levels</td>
<td>Decisions under certainty</td>
</tr>
<tr>
<td>(10) Hierarchy level</td>
<td>Operational (control)</td>
</tr>
</tbody>
</table>

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 \footnote{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 \footnote{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 \footnote{and, e.g., (Fowler, 2009; Suryanarayana et al., 2014).}
### Table 2: Phases of decision-making with relevant process input and output as well as characteristics for three selected key decision problems in refactoring.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Whether (and when) to refactor?</th>
<th>What to refactor (first)?</th>
<th>How to refactor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem recognition</td>
<td>How to identify/measure technical debt?</td>
<td>How to remove the system?</td>
<td>How to identify options?</td>
</tr>
<tr>
<td>Decision-making</td>
<td>How to analyze the problem context?</td>
<td>How to prioritize refactoring candidates?</td>
<td>How to compare options?</td>
</tr>
<tr>
<td>Evaluation</td>
<td>How to measure ROI?</td>
<td>How to evaluate the effects of refactoring regarding software design?</td>
<td>How to evaluate the effects of the refactoring regarding software behavior?</td>
</tr>
<tr>
<td>Process input</td>
<td>Software project</td>
<td>Resources (time frame, tools, developers etc.)</td>
<td>Measurable (time frame, tools, developers etc.), refactoring candidates</td>
</tr>
<tr>
<td>Process output</td>
<td>Allocated resources (down)</td>
<td>List of candidates (down), Refactored system with intended software design (up)</td>
<td>Refactored system with correct behavior (up)</td>
</tr>
<tr>
<td>Information needs</td>
<td>Budget, release plan, project context and state (condensed information)</td>
<td>Rules for candidate identification, prioritization paradigm (detailed information)</td>
<td>Rules for performing refactoring (detailed information)</td>
</tr>
</tbody>
</table>

### Findings

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.

### 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 viewpoint 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 (Grüning 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.

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 (Grüning and Kühn, 2013).
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 subprocesses, 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 subprocesses 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 subprocesses). 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 subprocess(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 A5 in Fig. 2 and for evaluating whether an error has been introduced (see step A2).
- **the difficulty** to perform the refactoring which is relevant for step A3 in Fig. 2 as well as the preceding steps which focus on identifying (step A2), comparing (step A4), and selecting refactoring option (step A5).
- **unclear ROI**, on management levels in steps A1 and A4 in Fig. 2 but also on operational levels, e.g. step A6.
- **technical issues**, e.g., as a lack of technologies or tools (relevant in many steps, e.g., step A2 in Fig. 2).
- **constraints set by the management** which are again result of steps A2 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; Fernandez 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 (Pridej, 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 A3 in Fig. 2; refactoring recommendation tools also A3, A4 and A5.
- **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 A6). 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 A3 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 A4 and A5, and partially A9 in Fig.2) (regarding the estimation of person-hours needed).

- **Automated Regression Testing Frameworks** (such as XUnit 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 C9).

- **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 A9 in Fig.2).
  - identifying smell false positives, see, e.g., (Fontana et al., 2016) (addressing B9),
  - paradigms for prioritizing candidates, see, e.g., (Ribeiro et al., 2016) (addressing step C9),
  - comparing and performing refactoring steps (also see e.g. (Fowler, 2009; Suryanarayana et al., 2014), addressing step C9).

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., B8, C11). On the other hand it becomes evident that some steps are not covered sufficiently by corresponding refactoring support techniques (see, e.g., A9).

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

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 2.4). 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).
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


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


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


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


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


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