Fakultät für Informatik
Technische Universität München

Continuous Research & Development

Scientific Research as Decision Support for Continuous Software Engineering in Domains with High Uncertainty

Simon Valentin Sebastian Klepper

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Hans Michael Gerndt

Prüfende der Dissertation:
1. Prof. Dr. Bernd Brügge
2. Prof. Dr. Helmut Krcmar

Die Dissertation wurde am 21.02.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 09.07.2022 angenommen.

Abstract

This dissertation focuses on improving decision-making in the context of Continuous Software Engineering (CSE). CSE is a collection of techniques, processes, and tools that emerged from the methodical and technological advances of the past decades. It combines iterative, incremental, and adaptive development in a holistic approach across processes in business, development, and operations.

Challenges arise when CSE is applied by projects under tight constraints, in domains with high degree of uncertainty, with decision-makers subject to cognitive biases. In these situations, pure empiricism might fail to optimize resource allocation, recognize risk to make it manageable, and ensure rational decision-making. As a result, organizations revert to planning activities in an attempt to counter the lack of effective decision-making, even after having achieved efficiency gains which are thereby nullified.

This anti-pattern motivates research to improve decision-making in CSE under these conditions. Emerging activities such as Continuous Planning, Continuous Innovation, and Continuous Experimentation already introduce new approaches and techniques. However, literature review and a systematic mapping study reveal gaps: disconnect between research, decision-making, and execution; lack of scientific research methodology; insufficient balance between predictiveness and opportunism.

This dissertation defines "Continuous Research & Development" (Continuous R&D) as an extension to CSE and introduces a corresponding model, process framework, and decision support system. The term indicates that Continuous R&D imports the scientific problem-solving approach from classical research and development into the context of CSE. Continuous R&D Model (CRDM) is a model for Continuous R&D describing entities and events.

CORTEX[1] is a process framework describing roles, activities, and artifacts in Continuous R&D. It applies strategies for continuously exploring problem and solution space, capturing uncertainty and identifying need for research, conducting research and capturing insights, and finally identifying actionable opportunities. The process has been verified in a simulation experiment to ensure completeness and correctness. Additionally, it was validated in 10 projects with a total of 70 participants and 3.920 person days runtime. Results show that benefits of "classic" R&D, such as scientific research and decision-making, can be carried over to CSE without impairing efficiency or quality.

RADAR[2] is a toolkit for knowledge management and decision support that enables real-time collaboration. Analysts can capture problems and goals, solution approaches, uncertainty, and insights. Decision-makers can discover need for research or actionable opportunities in a semi-automated process supported by visualizations. Validation was

[1] COntinuous Research and developmenT through EXperimentation
[2] Risk-Aware Development based on Agility and Research

done in 6 projects with a total of 20 participants and 670 person days runtime. The results show that tool support provided by RADAR improves the adoption of the CORTEX process.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Verbesserung von Entscheidungsfindung innerhalb von kontinuierlicher Softwareentwicklung (Continuous Software Engineering, CSE). CSE ist eine Sammlung von Techniken, Prozessen und Werkzeugen, die aus den methodischen und technologischen Fortschritten der letzten Jahrzehnte entstanden ist. Es kombiniert iterative, inkrementelle und adaptive Entwicklung in einem ganzheitlichen Ansatz über Geschäfts-, Entwicklungs- und Betriebsprozesse hinweg.

Herausforderungen ergeben sich, wenn CSE von Projekten unter starken Einschränkungen, in Domänen mit hohem Unsicherheitsgrad oder mit für kognitive Verzerrungen anfälligen Entscheidungsträgern angewandt wird. In solchen Situationen mag reine Empirie es nicht leisten eine Ressourcenverteilung zu optimieren, Risiken zu erkennen und somit kontrollierbar zu machen oder rationale Entscheidungsfindung zu gewährleisten. Als Folge versuchen Organisationen, den Mangel an effektiver Entscheidungsfindung, indem sie sich wieder Planungsaktivitäten zuwenden. Dies ist selbst dann der Fall, wenn hierdurch zuvor erreichte Effizienzsteigerungen wieder zunichte gemacht werden.

Dieses Anti-Pattern motiviert das Forschungsvorhaben, den Entscheidungsprozess innerhalb von CSE unter derartigen Bedingungen zu verbessern. Neu entstehende Aktivitäten wie kontinuierliche Planung (Continuous Planning), kontinuierliche Innovation (Continuous Innovation) und kontinuierliches Experimentieren (Continuous Experimentation) führen bereits neue Herangehensweisen und Vorgehensweisen ein. Allerdings zeigt eine Literaturrecherche sowie eine systematische Mapping-Studie Lücken auf: Entkopplung zwischen Forschung, Entscheidungsfindung und Ausführung; Mangel an wissenschaftlicher Forschungsmethodik; unzureichende Balance zwischen vorausschauender Handlungsweise und Opportunismus.

Diese Dissertation betrachtet „kontinuierliche Forschung & Entwicklung“ (Continuous Research & Development) als eine Erweiterung von CSE und führt ein korrespondierendes Modell, Prozess-Framework und System zur Entscheidungsunterstützung. Die Bezeichnung zeigt an, dass Continuous R&D das wissenschaftliche Problemlösen aus der klassischen Forschung und Entwicklung in den Kontext von CSE importiert. Continuous R&D Model (CRDM) ist ein Modell für Continuous R&D das Entitäten und Ereignisse beschreibt.

CORTEX[1] ist ein Prozess-Framework, das Rollen, Aktivitäten und Artefakte in CRD beschreibt. Es verwendet Strategien zur kontinuierlichen Exploration von Problem- und Lösungsraum, Erfassung von Unsicherheit und Identifikation von Forschungsbedarf und letztendlich Erkennen vorteilhafter Handlungsmöglichkeiten. Der zugrundeliegende Prozess wurde in einem Simulationsexperiment verifiziert, um seine Vollständigkeit

[1] COntinuous Research and developmenT through EXperimentation

und Korrektheit sicherzustellen. Zusätzlich wurde es in 10 Projekten mit insgesamt 70 Teilnehmern und einer Laufzeit von 3.920 Personentagen validiert. Die Ergebnisse zeigen, dass die Vorteile von „klassischer“ Forschung & Entwicklung, wie beispielsweise wissenschaftliche Forschung und Entscheidungsfindung, auf CSE übertragen werden können, ohne dass dabei die Effizienz oder Qualität des Projekts leidet.

RADAR[2] ist ein Werkzeug für Wissensmanagement und Entscheidungsunterstützung, das auch Zusammenarbeit in Echtzeit ermöglicht. Analysten können Probleme und Ziele ebenso erfassen wie Lösungsansätze, Unsicherheit und Erkenntnisse. Entscheidungsträger können Forschungsbedarf entdecken oder Handlungsmöglichkeiten erkennen. Dies geschieht in einem halbautomatischen Prozess, der durch Visualisierungen unterstützt wird. Die Validierung erfolgte in 6 Projekten mit insgesamt 20 Teilnehmern und einer Laufzeit von 670 Personentagen. Die Ergebnisse zeigen, dass die Unterstützung durch das RADAR-Werkzeug die Annahme des CORTEX-Prozesses verbessert.

[2]Risk-Aware Development based on Agility and Research

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance by many people. I want to explicitly thank them acknowledge their support.

First, I would like to express my deep gratitude to my supervisor, Professor Bernd Brügge, whose expertise and guidance was invaluable and provided the foundation for this research. I am thankful for the openness, trust, patience, inspiration, and encouragement. His insightful feedback pushed me to reflect on my goals, broaden my horizon, sharpen my thinking, and constantly improve my work.

I would also like to thank Thomas von Chossy who inspired my to pursue this academic goal in the first place and was always there as an advisor, coach, and motivator.

I thank Sebastian Jonas for his support as the mentor of my dissertation but also as a colleague over many years in different environment. I also thank Stephan Krusche for his advisory over the years, which started with my Master's thesis which was the first step in this direction.

I want to thank the contributors to research and publications surrounding this dissertation for their effort and dedication: Christian Grimm, Anton Widera, Fabiola Moyon, Özge Soydemir, Julia Ludmann, Andre Müller.

Particular thanks are due to my esteemed colleagues at the Chair for Applied Software Engineering for always offering a sympathetic ear, valuable feedback and advice, or a helping hand. I especially want to thank Lukas Alperowitz, Jan Ole Johanßen, Stefan Nosovic, Damir Ismailović, Nadine von Frankenberg, Florian Bodleé. Special thanks to Leon von Tippelskirch for a long series of successful and insightful projects in the iPraktikum capstone course.

I would also like to express my gratitude to Monika Markl, Helma Schneider, and Uta Weber for the uniquely supportive environment they created.

I thank my friends and colleagues outside of the chair for their continuous support, helpful discussions and feedback, as all as fun and encouragement.

Finally, I want to express my love and gratitude to my family and my partner Nina. This endeavor would not have been possible without your understanding and support.

Contents

Conventions

Style This dissertation uses American English, the only exception being direct quotes. The singular "they" (including the related forms them, their, and themselves) serves as the generic third-person pronoun.

Trademarks Company names as well as product names are understood to be registered trademarks. Whenever possible, footnotes provide links to more information.

UML UML diagrams use UML 2.0 notation. Multiplicity is irrelevant or can be inferred from context if not noted explicitly. However, associations can be assumed to have `1:1` and aggregations as well as compositions to have `1:*`. Class diagrams may use colored clusters to represent packages.

Departing from UML conventions, entity names such as packages and classes in UML class diagrams are written with spaces for readability, e.g., `Variation Point` instead of `VariationPoint`. However, class names in arguments of class methods remain written without spaces for clarity, e.g, `cover(points: VariationPoint[])`.

1 Introduction

This dissertation focuses on the importance and improvement of decision-making in the context of Continuous Software Engineering (CSE) [1]. In CSE, activities from the domains of business, development, and operations are closely tied together and executed in a continuous fashion. Therefore, decisions made in one specific area of the overall process have the potential to significantly impact the overall project. This is why the quality of these decisions comprises the center of this research, along with potential methods to measure and positively influence it.

Our research is primarily motivated by trends in software engineering of moving towards more continuous and integrated approaches, both methodically and technologically, and the possibilities they open up for more advanced techniques. Secondarily, the difficulties that projects are facing suggest that there is an underlying problem with decision-making that the empirical nature of modern methodologies cannot sufficiently address. The following paragraphs provide more details on these issues.

The remainder of this chapter is organized as follows: Section 1.1 provides details about the CSE environment, decision-making as a success factor for software projects, and the most important impediments in this context: Uncertainty, constraints, and cognitive bias. Section 1.2 derives corresponding research objectives, divided into goals, problems, and questions. Section 1.3 describes the methodology used for the overall research project as well as for treatment design, empirical research, and inference. Section 1.4 lays out the structure of this dissertation.

Trends in Software Engineering Methodology Continuous Software Engineering emerged in the industry of software-intensive systems in response to competitive pressure. After the traditional linear development model (e.g., waterfall [2] and V-Modell [3]) had been replaced by iterative models like Rational Unified Process (RUP) [4] as well as V-Modell XT [3], agile practices emerged such as Extreme Programming (XP) [5] and Scrum [6]. These process models not only introduced incremental development but also emphasized an adaptive as opposed to a plan-driven approach. Meanwhile, techniques from lean manufacturing such as Kanban introduced principles for moving from sequential to continuous execution of activities [7]. Advanced technology and techniques allowed continuous integration and delivery of software [8]. Finally, Fitzgerald and Stol [9] identify trends to integrate between important software engineering activities, most notably between all of development and operations ("DevOps") as well as business and development ("BizDev").

Road to Continuous Software Engineering Continuous Software Engineering (CSE) aims to take advantage of these advancements and transform software engineering from traditional development to an innovation system, enabling organizations to compete in a

modern marketplace [10]. Most importantly, agile and lean methodologies are applied to transform the development organization from linear "production" towards continuous research and development. Bosch [10] describes this transformation in the "Stairway to Heaven" model as multiple steps leading from traditional development to an innovation system. The model defines the following key stages along this path:

1. traditional development: slow, infrequent releases of big changes;
2. R&D organization all agile: adoption of Agile practices, more and smaller changes;
3. Continuous Integration: automated testing of every change;
4. Continuous Deployment: more frequent releases through better quality control;
5. R&D as an innovation system: experiments and optimizations in frequent releases.

Although the model focuses on the way that software is produced and delivered, it already emphasizes the important difference between a "production mindset" and a "learning mindset". This difference is of particular importance for facilitating innovation.

Anti-Patterns in Continuous Software Engineering CSE already handles uncertainty insofar as each change is continuously delivered and feedback is collected from operations. Thus, bad decisions can be recognized and corrected as necessary. However, this retroactive compensation becomes increasingly cumbersome and wasteful—especially if the project is subject to strong constraints and cognitive biases interfere with decisions. To counterbalance this, companies increasingly transition to preemptive analysis and planning. This, however, is a step backwards to linear research and development and incompatible with the aspiration of CSE to execute all activities in a continuous flow. Empirical evidence from industry shows this works, but is analogous to setting wrong course and then slowly correcting it. Companies try to counteract with analysis and planning, risking to break continuity[1]. Ironically, efficiently executing bad decisions can be just as bad as "analysis paralysis". Bad decisions can therefore be considered unnecessary mistakes that lead to waste, both of which should be avoided according to Lean principles.

[1] VersionOne Inc. *10th Annual State of Agile Report*. 2016. URL: https://explore.versionone.com/state-of-agile/versionone-10th-annual-state-of-agile-report-2, VersionOne Inc. *11th Annual State of Agile Report*. 2017. URL: https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2, VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

1.1 Problem Context

The demands for software engineering in complex domains have evolved far beyond linear approaches based on planning. For instance, so-called "ultra large scale" software systems (ULS) require dealing with their sheer size, solving wicked problems, satisfying multiple customer groups with different needs simultaneously and making correct decisions while dealing with risks arising from changes [14]. Quality products need to be delivered under pressures like reduced time-to-market, unpredictable market needs as well as complex and evolving customer needs [10].

Adopting a plan-based, sequential approach to software engineering yields risks due to slow development cycles, lack of flexibility, and disconnects between individual development activities [15, 10, 1]. Traditional methods assume that everything about a project can be known beforehand and thus a complete, consistent and correct set of requirements can be specified [16, 17, 18].

Agile methodology provides empirical process models to increase flexibility and responsiveness, emphasize customer collaboration, enhance creativity, reduce uncertainty, and provide efficient change management [19, 16, 20, 21, 22]. Instead of trying to know everything beforehand, changes and unforeseen events are embraced as a chance to improve the quality of a solution [23]. This enables software projects to better cope with fast-changing, unpredictable market needs [10].

As a result, many companies have adopted agile methods like extreme programming (XP) or Scrum [20]. However, applicability may suffer for instance in larger projects or companies with difficult communication structures or multiple interdependencies. This may lead to some formality being reintroduced at the cost of flexibility [20]. Frameworks like SAFe[2] aim to enable agile concepts in larger enterprises, but may also suffer from standard problems arising through increased scale[3]. However, benefits like reduced risk, faster development cycles and dealing with unclear requirements warrant at least the adoption of agile concepts [20, 21].

While agile methods mostly focus on the development of software and reducing risks of change, continuous software engineering and lean principles emphasize a continuous end-to-end flow of activities [1]. Additionally, to achieve innovation in complex environments and systems, chaordic ways of working aim to enhance learning and creativity through balancing defined processes (order) with empirical, nondeterministic processes (chaos) [22].

1.1.1 Continuous Software Engineering

Continuous Software Engineering (CSE) describes the collection of activities necessary for software product development with a focus on integration between activities as well as principles from Agile development and Lean management [9]. CSE is rooted in the Continuous Integration and Continuous Delivery [8] as well as the Lean Startup movement focusing on experimentation and value creation [15]. It extends the purpose of software

[2] Scaled Agile Framework. URL: `https://www.scaledagileframework.com/`

[3] see Prowareness. *Ist das Scaled Agile Framework (SAFe) wirklich agil?* 2015. URL: `https://www.scrum.de/ist-das-scaled-agile-framework-safe-wirklich-agil/`.

engineering to acting as an innovation system for the organization [25]. This can be seen as a superclass of all activities in software engineering.

Bosch [10] describes a maturity model for CSE organizations with "R&D as innovation system" at the highest level. To reach this point, organizations face the challenges of establishing reactive capabilities through Agile development techniques, integrating activities to form a feedback loop, and using Lean management techniques actually work in a continuous fashion[4]. Figure 1.1 depicts the feedback loop provided by Continuous Experimentation and Continuous Innovation and the critical links between business and development ("BizDev") as well as between development and operations ("DevOps"). Fitzgerald and Stol [1] and Fitzgerald and Stol [9] identify a gap in the place of BizDev which is consistent with industry studies reporting that organizations struggle to find product-market fit despite employing development methodologies (Agile, Lean, DevOps). This dissertation targets the Continuous Experimentation & Innovation loop with a focus on the integration between decision making processes in business with implementation processes in development.

Challenges with Continuous Innovation Providing continuous innovation means discovering customer needs, potential opportunities and gaps in evolving markets well before they arise [1]. It involves dealing with complex problem domains, dynamic markets as well as changing or conflicting customer groups [14, 10]. Agile and continuous methods enable flexibility towards changes and to efficiently coordinate development efforts. However, validating the correctness of a solution, reducing project risks or improving overall product quality still remains an issue to software creators even today. To still maintain development speed while dealing with these issues, it is required to focus efforts on value bringing activities and eliminate wasteful tasks [7, 26].

To mitigate these risks and reduce uncertainty, continuous experimentation arises as a measure for increasing creativity, enabling continuous learning, supporting data-driven decision making, predicting product value, and closing the "open loop" problem between product management and development [27, 28, 29, 22]. The latter refers to the need to gather accurate and timely feedback from customers to employ within strategic product decisions [27]. Continuous software engineering sets its focus on frequently performing and shortening customer feedback loops in order to discover customer needs and behavior [10]. Thus it enables innovation drivers to quickly design, execute and evaluate cheap experimentation efforts in order to gain insight.

Challenges with Continuous Experimentation However, enabling decision support, continuous practices and setting up continuous experimentation still poses many challenges [10, 30, 31]. First and foremost, an experimentation environment has to be set up where massive amounts of data can be continuously gathered, stored and evaluated for information. This can be difficult or even impossible to set up depending on time, technical, organizational or other limitations [30]. Experimentation often requires some

[4] It's worth noting that "continuous" is a misnomer in the scientific sense because software engineering process models that use the term work only in discrete steps.

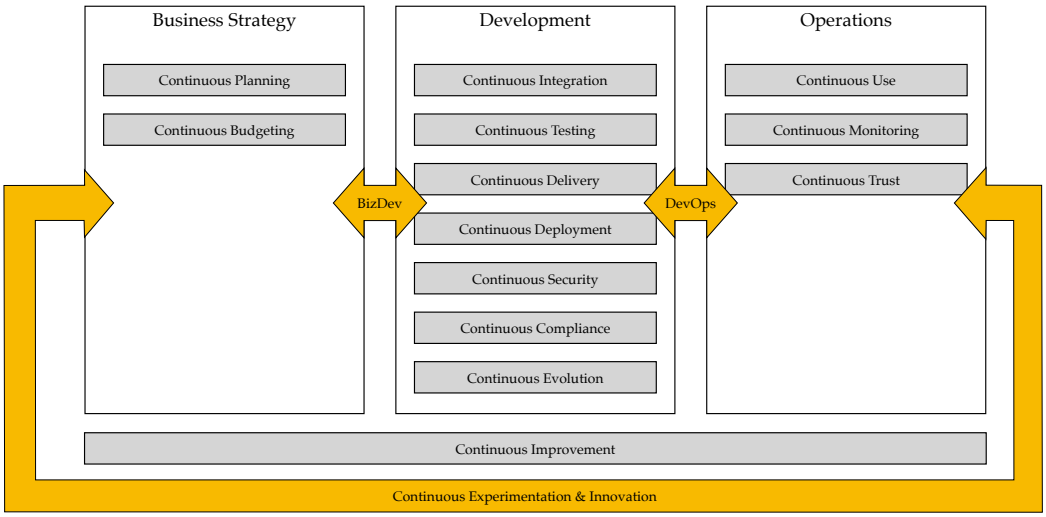


Figure 1.1: Illustration of the bidirectional connections between business strategy, development, and operations (adapted from the Continuous* model by Fitzgerald and Stol [1]).

kind of live product to achieve validation through deployment [10], which might not always be feasible. When setting up experimentation in large organizations, often a fundamental conflict with the established way of thinking and adopting an experimental, creative mindset on both developer and customer side becomes evident [30, 31].

Furthermore, a development team that has never worked within an experimental approach before may have trouble adopting this technique [30]. In this context, experimentation models are often too abstract or require presumed knowledge to be efficiently deployable [32, 25, 33, 27, 34, 35, 28, 29]. Their main focus lies on either the initial exploration, depicting the overall experimentation process covering multiple parts of continuous software engineering at once, or on feature refinement opposed to achieving innovation. Additionally, there is a lack of details how to perform certain activities. For instance, in terms of hypothesis generation and evaluation, it is only superficially stated how to generate artifacts to put under test or how to enable decision support from hypothesis testing [29]. These premises are making it difficult to teach these approaches to newcomers to experimental development.

1.1.2 Decision Making and Decision Support

The overall focus is the successfulness of software product development in domains with high degree of uncertainty—the goal being, of course, to increase it.

When looking at success factors, it is important to differentiate *efficiency* ("doing things right") from *effectiveness* ("doing the right things"). In software product development,

efficiency is a matter of execution whereas effectiveness is a matter of decision-making [see 36, 37].

While efficient execution is an integral part of CSE, especially through the application of lean principles, concerns about effective decision-making seem limited to planning activities and not integrated throughout the process in the same way [see 1].

Taking a closer look at the role of decision-making in software engineering, it becomes apparent that it is vital for the purpose of solving problems: On the one hand, the solution needs to be found in the problem space (or problem domain) which requires problem analysis as well as discovery and evaluation of available solution; on the other hand, the solution needs to be created in the solution space (or solution domain) which requires technical analysis and careful consideration of tradeoffs [38, 39, 36, 40].

At the boundary between problem space and solution space are *requirements* and their quality plays a central role in the success of the problem-solving endeavor. More precisely, traits of good requirements are the "4 Cs of requirements": completeness, consistency, correctness, clarity [36, 37, 17, 18]. The aim therefore is to improve the decision-making processes in requirements engineering to produce better requirements.

The following paragraphs elaborate the issues caused by **uncertainty** (the inability to assess the outcome of decisions) and explore challenges with **project constraints** (limits to the ability of preparing and executing decisions) as well as the potential impact of **cognitive bias** (reduced rationality when making or evaluating decisions). Furthermore, they discuss the difference between **decision-making** and **decision support**.

Challenges through Uncertainty One issue of particular interest for decision-making is **uncertainty**, i.e., the inability to assess the outcome of decisions [see 41]. It has implications for risk management but is different from risk: while uncertainty implies a potential, unpredictable, and uncontrollable outcome, risk is a consequence of action taken *in spite* of uncertainty [42, 43]. This means decisions under uncertainty carry unknown risk that is hard, if not impossible, to manage. The pressing question here is how software product teams can continuously and efficiently reduce uncertainty to make decisions less risky.

Examples of uncertainty in software product development would be a lack of domain knowledge in all relevant fields or the unpredictability of complex environments. These issues may result in decisions that later turn out to be wrong and negatively impact for the project.

According to Chapman and Ward [44] uncertainties during the project life cycle are particularly evident in the pre-execution stages, when they contribute to uncertainty in five following areas:

1. Variability associated with estimates
2. Uncertainty about the basis of estimates
3. Uncertainty about design and logistics
4. Uncertainty about objectives and priorities

5. Uncertainty about fundamental relationships between project parties

These areas of uncertainties are defined as important, but generally they become more important throughout the list.

Challenges through Project Constraints Constraints in decision-making can be understood as conditions of an optimization problem that the solution must satisfy. More precisely, constraints are requirements to the variables of a cost or reward function. They can be "hard constraints" (mandatory satisfaction) or "soft constraints" (variable satisfaction).

In (software) project management there are traditional constraints of time, scope, and resources. This definition was later updated with agility and quality [45]. Projects might work on a large scope or have a high degree of uncertainty about their scope. Their stakeholders might have high expectations regarding the speed of development as well as the quality (performance, safety, robustness) of delivered solutions. However, they might be working with a small team and budget while having to constantly keep their options open to react to new technological changes.

CSE projects are particularly likely to encounter these constraints because they consistently are the very reason why organizations adopt agile and lean methodologies [46, 47, 48, 49, 50, 51, 52, 53, 54, 11, 12, 13, 55].

Decisions that don't consider these constraints appropriately risk being wasteful or even exceeding the project's boundaries. Decision makers therefore need to consider all relevant constraints not only while executing on decisions but also with actions to improve decision-making. In this context, one important challenge for balancing constraints in requirements engineering is weighing the risk associated with decisions agains the effort necessary to reduce the uncertainty.

Challenges through Cognitive Bias Human decision making often shows systematic mistakes caused by shortcuts in the way decisions are made; these patterns of deviation from norm or rationality in judgment are called cognitive biases [see 56]. More precisely: A single bad decision is a mistake; systematic (repeated and unrecognized) bad decisions are errors; bias is a distortion of perception that causes errors.

For example, project managers might be subject to the "planning fallacy" or "optimism bias", causing them to constantly underestimate the amount of time required to complete a task. While researching information for decisions, they might fall victim to the "confirmation bias" and only consider information that confirms their preconceived views. When faced with negative outcomes, they might refuse to correct course due to the "sunk cost fallacy" which leads them to assume that previously invested resources would be wasted.

There are many more cognitive biases relevant in this context that not only occasionally lead to bad decisions but fundamentally shape the course of action (e.g., ambiguity aversion and risk aversion) and can even defeat the "correction mechanism" of agile process models which assume rational evaluation of feedback [see 32]. Identifying and

avoiding cognitive bias both when making and evaluating decisions is therefore considered a key aspect to effective and successful decision-making.

Decision-Making vs. Decision Support As seen in the introduction, decision making is a crucial activity in the software development process. However, searching for the ideal model and the optimal solution belonging to it, often ends in a deadlock, because the problem domain is not understood properly [57]. This argument is the cornerstone of the distinction between decision making and decision support [41]. The main difference is that decision making tries to always find the optimal solution which often does not exist. In complex problems for example an optimal solution cannot be found. Decision support on the other hand realizes that such an optimal solution does not exist and supports the decision maker in the decision process by helping him to find a satisfactory solution for the actual problem [41].

Different decision problems are a well-researched field in decision science. Roy and Bouyssou [58] discern between the tree categories of selection, triage and ranking [see 41]. Besides these categories, Simon [59] differentiates between structured and unstructured decision problems. There is a clear solution in how to solve a structured problem, i.e. applying best practice. In an unstructured problem, there exists a lot of uncertainty in the problem domain and information or knowledge is not available in the extent needed. As a result, these problems are especially hard to solve.

To tackle decision problems, different approaches for decision support were developed, like IBIS [60] and PANDA [61], both of which will subsequently be presented in detail. IBIS is a sociological approach to decision support suited for wicked problems. The PANDA workflow is designed for empirical product development and supports in approaching complex problem domains. However, it does not explicitly specify its information objects and how they should be stored.

Specifying how information should be stored belongs to the field of knowledge management. There exist multiple definitions for knowledge management [62] which makes is hard to find one unifying definition. Most definitions include the transformation of information into knowledge which is made available to the people who want to apply it [62]. In this thesis this would correspond to all actors involved in the decision process.

1.2 Objectives and Contributions

This dissertation introduces “Continuous Research & Development” (Continuous R&D) as an extension to CSE and introduces a corresponding model, process framework, and decision support system. Continuous R&D integrates the scientific problem-solving approach from classical research and development into the context of CSE using the design science approach by Wieringa [63]. When defining the goals of a design science research project, Wieringa distinguishes between the stakeholder goals and research goals. From the problem context, the CSE project team is the external stakeholder to this research project. Its goal is to work on the right tasks (effectiveness), use time and resources optimally (efficiency), and produce a result that fulfills its requirements (quality). The corresponding goal for the researcher is to find a method for improving the effectiveness of decision-making while not impacting the other success factors.

Stakeholder Goal 1 *CSE project teams want to optimize the success factors of effectiveness, efficiency and quality.*

Research Goal 1 *Improve decision-making in CSE projects using Continuous R&D.*

Figure 1.2 visualizes these goals as well as associated knowledge questions and design problems according to the contributions of this dissertation.

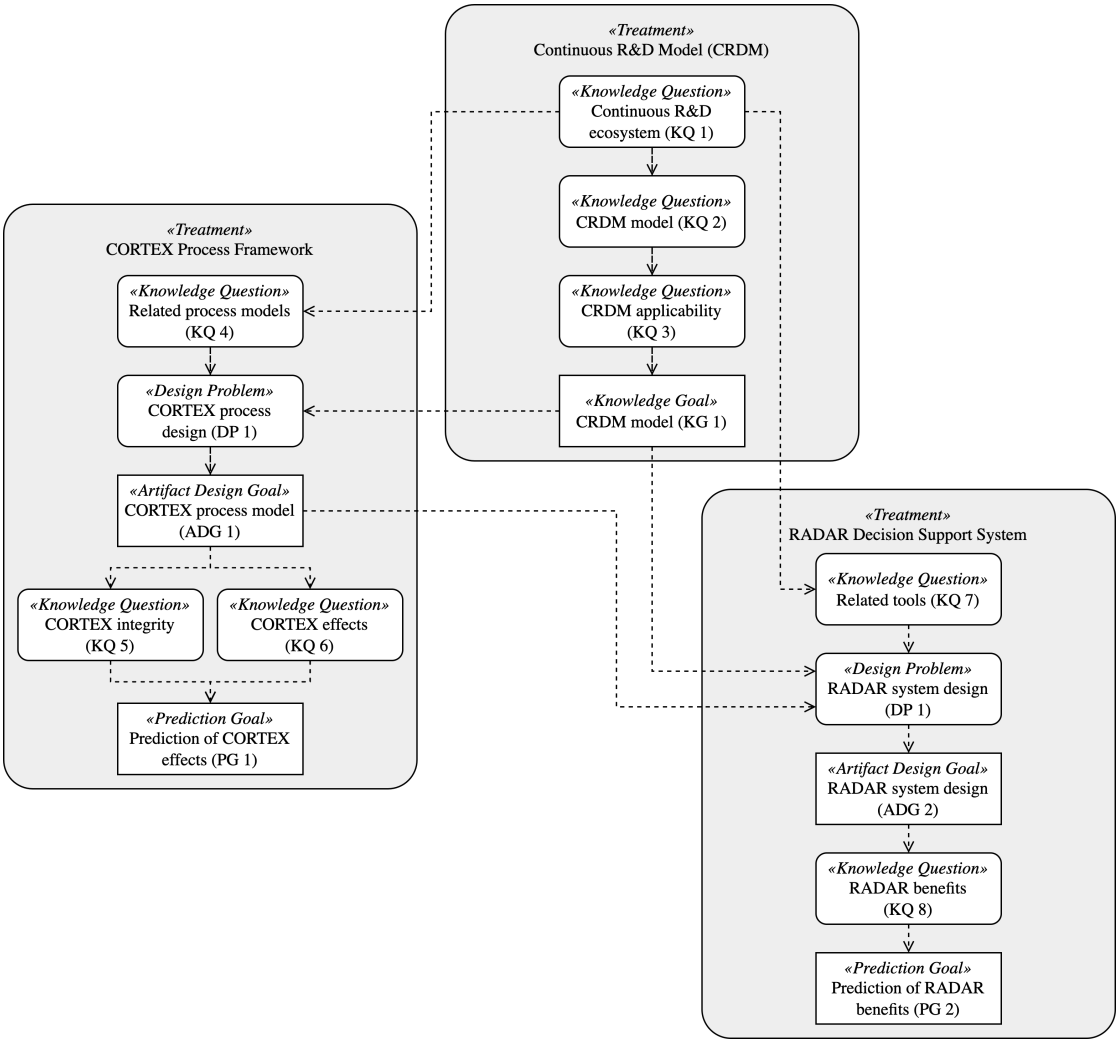


Figure 1.2: Structure of design science research objectives behind the CRDM model, CORTEX process framework, and RADAR decision support system.

The first contribution is CRDM, a model that allows to understand the problem context of Continuous R&D. This involves collecting knowledge about the ecosystem of

decision-making in CSE, formalizing that knowledge into a coherent model (Continuous R&D Model, CRDM), and validating the applicability of this model. In terms of Wieringa [63], this represents a knowledge goal to describe and explain phenomena in the problem context, followed by knowledge questions asking for corresponding insight.

Knowledge Goal 1 *Formulate a coherent model for scientific problem solving in CSE.*

Knowledge Question 1 *What requirements do processes and tools need to fulfill to suit the characteristics of decision-making in CSE?*

Knowledge Question 2 *Which concepts are central to scientific problem solving?*

Knowledge Question 3 *Is CRDM applicable in the context of Continuous R&D?*

The second contribution is CORTEX (COntinuous Research and developmenT through EXperimentation), a process framework that defines how to solve problems using research in the context described by CRDM. As a basis, related work needs to be reviewed in order to determine gaps in the status quo and references for process design. Then the process framework needs to be designed by integrating mechanisms of scientific problem solving with existing CSE processes. In terms of Wieringa [63], the design of CORTEX represents an artifact design goal, producing an artifact that fulfills a stakeholder goal. This defines a so-called design problem that describes the desired artifact and its desired effect.

Artifact Design Goal 1 *Design CORTEX, a process framework for scientific problem solving in CSE based on CRDM.*

Knowledge Question 4 *Which existing process models fulfill the requirements from the Continuous R&D ecosystem and could therefore serve as references for CORTEX?*

Design Problem 1 *How to design a decision-making process framework that integrates scientific problem solving with CSE so that decisions can be made more effectively?*

The process framework must then be validated before proceeding. Specifically, its behavioral integrity and characteristics when applied in real-world CSE projects need to be investigated before using it in the problem context or in subsequent design activities. In terms of Wieringa [63], the validation of CORTEX pursues a prediction goal, forecasting how an artifact will interact with a problem context.

Prediction Goal 1 *Predict the impact of using CORTEX in CSE projects.*

Knowledge Question 5 *Does CORTEX behave as expected in CSE projects?*

Knowledge Question 6 *How does CORTEX impact decision-making in CSE projects?*

The third contribution is RADAR (Risk-Aware Development based on Agility and Research), a system for knowledge management and decision support. The design process is similar to that of CORTEX but with a focus on system design instead of process design. Existing tools for knowledge management and decision support need to be reviewed in order to find a starting point for system design. Then RADAR needs to be designed either on the basis of existing tools or completely from scratch. Again, this represents an artifact design goal and a corresponding design problem.

Artifact Design Goal 2 *Design RADAR, a decision support system for CORTEX based on CRDM.*

Design Problem 2 *How to design a decision support system that captures knowledge and uncertainty so that CORTEX can be applied more easily?*

Knowledge Question 7 *Which existing tools fulfill the requirements from the Continuous R&D ecosystem and could therefore serve as the basis for RADAR?*

The RADAR decision support system must be validated as well. Specifically, its benefits for applying each aspect of CORTEX in CSE projects need to be investigated. Again, this represents a prediction goal with a corresponding knowledge question.

Prediction Goal 2 *Predict the benefits of RADAR when applying CORTEX.*

Knowledge Question 8 *Does RADAR help with applying each aspect of CORTEX?*

1.3 Research Approach

In Wieringa's terminology, this dissertation is a design science research project with three design cycles, one each for CRDM, CORTEX, and RADAR. Wieringa [63] decomposes the overall design science endeavour into iterations of *designing* a treatment and *investigating* its effects. Within this overarching **engineering cycle** (1–5), the task of designing the treatment further consists of investigating the problem (1), designing a treatment (2), and validating it (3). This is called the **design cycle**. Subsequently, the validated treatment is implemented (4) and evaluated (5) in the real world.

1. **Problem investigation:** Investigate real-world problem context before designing a treatment.

2. **Treatment design:** Design a treatment for investigated problem.

3. **Treatment validation:** Validate that treatment fulfills requirements before implementation. This might also include real-world validation.

4. **Treatment implementation:** Implement treatment in real-world problem context.

5. **Implementation evaluation:** Evaluate effects of treatment when applied in the real-world problem context.

Within the engineering cycle, Wieringa makes a distinction between real-world and laboratory conditions. Problem investigation and implementation evaluation happen in the real world problem context. Treatment design, treatment implementation, and treatment validation happen in a laboratory environment. While real-world interactions with the problem context provide actual, tangible evidence for further research, the laboratory environment provides controlled conditions in which to design, validate, and implement a suitable treatment. Wieringa [63] states that a design science research project only covers the design cycle, i.e., problem investigation, treatment design, and treatment validation. A research project might even execute the design cycle multiple times to refine and extend artifacts or validation results. For the concretization of the research project, we use the object-oriented design methodology OOSE [36]. Figure 1.3 shows how the OOSE process is applied throughout the design science research project.

1.4 Outline of the Dissertation

Chapter 1 introduces the main motivation for this research as well as the corresponding approach used in this dissertation. Section 1.1 problem context within CSE, consisting of decision-making as a success factor on the one hand and impediments to decision-making on the other. Section 1.2 defines research goals, design problems, and knowledge questions to be pursued in the research project. Section 1.3 introduces suitable methodology for research, design, and validation activities. Section 1.4 (current section) defines the dissertation structure, describes the research approach for each activity, and further clarifies the scope of this dissertation.

Chapter 2 coins the term "continuous research & development" (Continuous R&D) for scientific problem solving in the context of continuous software engineering. Continuous R&D Model (CRDM) provides a coherent model for this practice. The model serves as the foundation for subsequent artifact design activities. Section 2.1 collects roles as well as use cases and describes the target environment of decision-making in CSE. Section 2.2 constructs a model for scientific decision making reaching from problem solving and decision-making over knowledge management and decision support to scientific

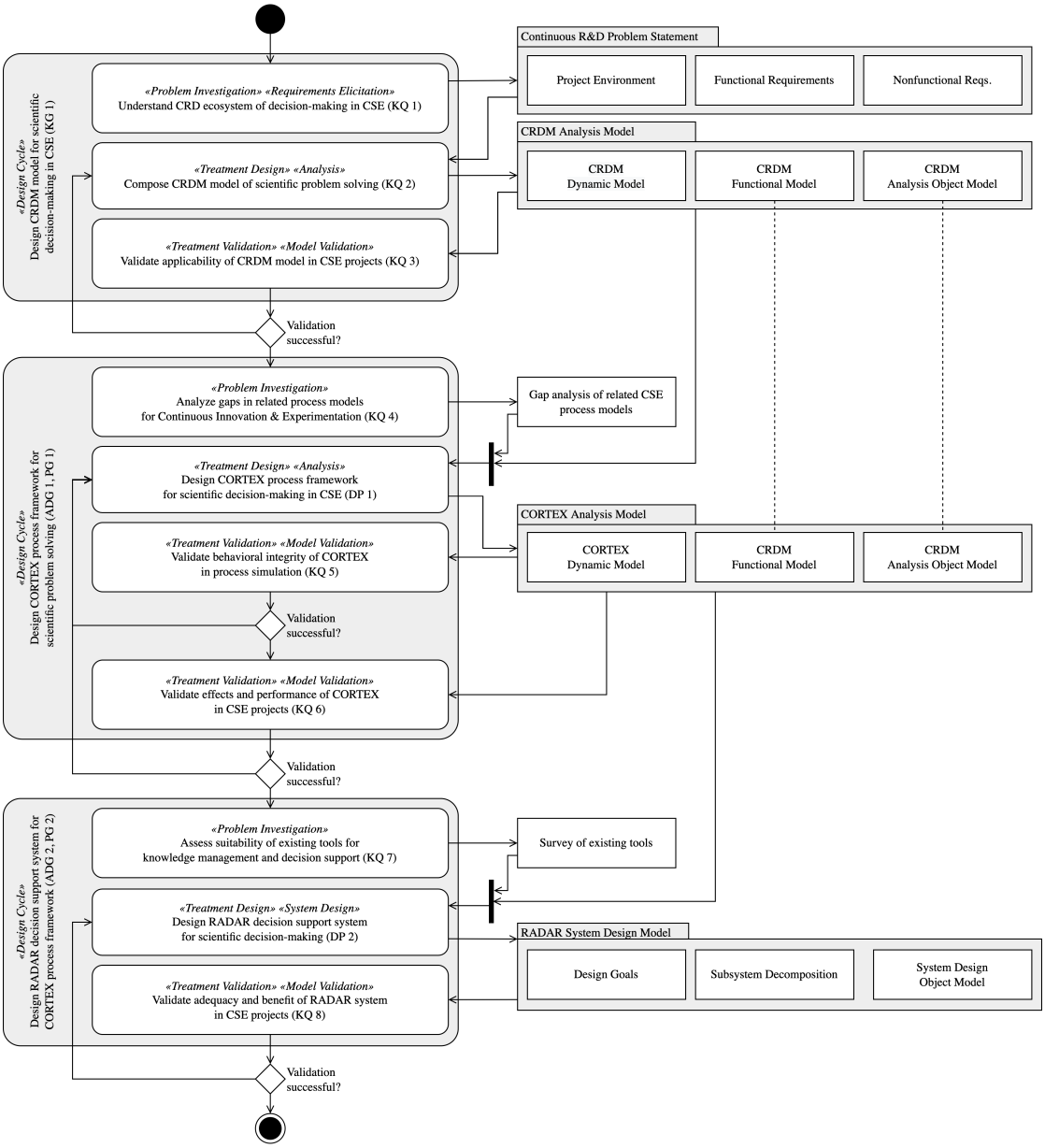


Figure 1.3: Overview of activities and entities in a tailored version of the OOSE process merged with design science methodology.

research and experimentation. Section 2.3 validates understandability and applicability of Continuous R&D as well as its implications for decision-making.

Chapter 3 designs CORTEX (Continuous Research and development Through Experimentation), a process framework for scientific problem solving in CSE based on the previously constructed model. The chapter also predicts the impact of using the CORTEX in CSE projects. Section 3.1 explores related process models for innovation and experimentation and investigates their suitability for Continuous R&D. **Section 3.2** designs the process framework based on an event-driven core lifecycle and a main problem solving workflow. It then defines sub-workflows for decision-making, problem and solution space exploration, and research. It also provides details on potential starting points for process tailoring. Section 3.3 validates the requirements satisfaction and behavioral integrity of CORTEX using a simulation. Section 3.4 validates the performance of CORTEX when applied in a real-world context, specifically its effects on decision effectiveness as well as side effects on quality and efficiency.

Chapter 4 designs RADAR (Risk-Aware Development based on Agility and Research), a decision support system for CORTEX. The chapter also predicts the benefits of RADAR for applying CORTEX. Section 4.1 explores related tools for knowledge management and decision support and investigates their suitability for applying CORTEX according to the previously derived requirements. Section 4.2 designs the decision support system by first defining a suitable subsystem decomposition. It then describes relevant aspects of persistent data management as well as access control and security. Finally, it defines behavior for global software control as well as in boundary conditions. Section 4.3 validates the adequacy and benefit of RADAR with regards to knowledge management as well as decision support.

Chapter 5 concludes this dissertation by summarizing its results and providing an outlook on potential future work. Section 5.1 aggregates the findings and contributions according to the research goals: CRDM and insights on its applicability for scientific problem solving in CSE, CORTEX and insights on its impact on effective decision-making, RADAR and insights on its benefits for decision support. Section 5.2 discusses threats to validity in the construction and validation of these artifacts in order to avoid drawing the wrong conclusions from results. Section 5.3 outlines areas for future work. For each artifact (CRDM model, CORTEX process framework, RADAR system) there is need for in-depth evaluation by continuing the design science engineering cycle. Likewise, all artifacts would benefit from expansion (adding more detail and precision) and extension (adding new elements and relationships).

Appendix A contains licenses for material used in this dissertation, such as permission for reuse for prior publications. Appendix B holds additional elements of CRDM. Appendix C provides extensive descriptions of the research activities surrounding CORTEX. Specifically, it describes the investigation of related process models, the verification using software simulation, and validation in real-world projects. Appendix D likewise provides background information on the investigation, design, and validation of RADAR. This concretely means details on the investigation of related tools as well as the prototypical implementation used in validation.

2 CRDM: A Model for Continuous Research & Development

The first design cycle of this dissertation pursues knowledge goal 1: Formulating a coherent model for scientific problem solving in CSE. The result is Continuous R&D Model (CRDM), consisting of a dynamic model, a functional model, and an object model. CRDM captures the essential concepts of Continuous R&D and serves as the basis for the subsequent treatment design of the CORTEX process framework and the RADAR decision support system.

Figure 2.1 visualizes how the combination of design science and OOSE methodologies addresses the knowledge goal by investigating its associated knowledge questions. Section 2.1 conducts the problem investigation phase of the design cycle, investigating the ecosystem of decision-making in CSE (knowledge question 1) using OOSE requirements elicitation, which produces the Problem Statement for Continuous R&D. Section 2.2 conducts the treatment design phase, modeling scientific decision-making (knowledge question 2) using OOSE analysis. This results in the CRDM Analysis Model. Section 2.3 conducts the treatment validation phase, examining the applicability of CRDM in multiple CSE projects (knowledge question 3). In OOSE terms, this represents model validation covering the Analysis Model using a prototypical implementation[1].

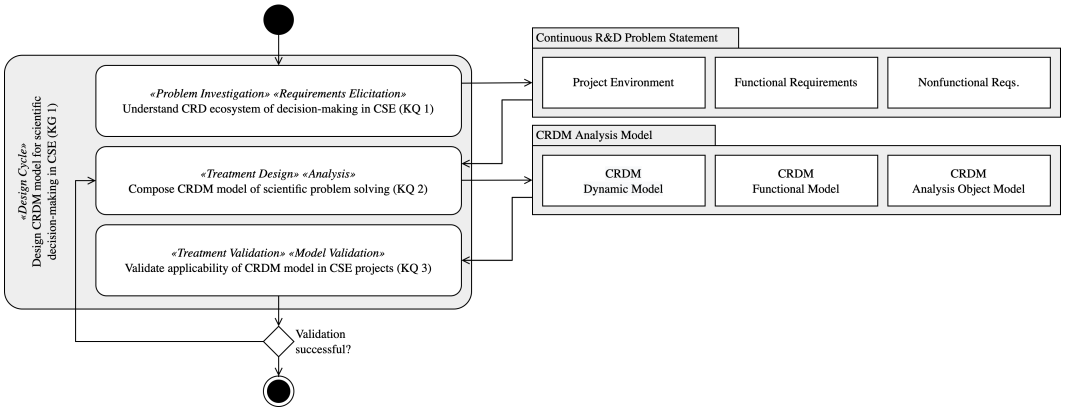


Figure 2.1: Overview of activities and entities in the first design cycle of the research project, yielding Continuous R&D Problem statement and CRDM Analysis Model.

[1]The term "implementation" is used in the software engineering sense, not in the design science sense.

2.1 Ecosystem of Decision-Making in CSE

Before a treatment to a problem can be sensibly designed, the problem and its context need to be sufficiently understood in their current status [63]. This section conducts problem investigation within the design cycle for CRDM. It investigates knowledge question 1: What requirements do processes and tools need to fulfill to suit the characteristics of decision-making in CSE? This analysis poses the following research questions:

1. What are the defining characteristics of CSE projects and which constraints does this pose when designing processes and systems for decision support?
2. What are the guiding activities for decision-making within Continuous Software Engineering?
3. Which additional aspects do stakeholders care about when preparing, supporting, making, and executing decisions?

Figure 2.2 depicts the process of literature review on the CSE ecosystem using the Comprehensive Literature Review (CLR) framework as described by Onwuegbuzie and Frels [64]. It represents the requirements elicitation activity in OOSE and produces the Continuous R&D Problem Statement which includes project environment (section 2.1.1), functional requirements (section 2.1.2), and nonfunctional requirements (section 2.1.3). The Continuous R&D ecosystem is analyzed through an integrative literature review and qualitative analysis to synthesize results [65].

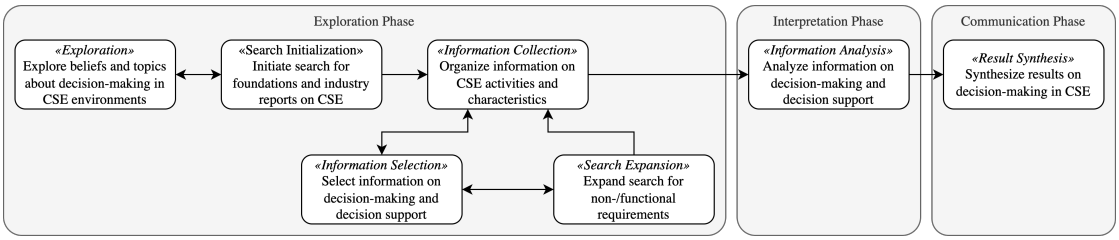


Figure 2.2: Process of literature review on the CSE ecosystem, based on the CLR framework [64].

2.1.1 Project Environment

Before going into detail on the activities in CSE, it is necessary to understand its foundational principles. These principles govern each aspect of CSE and therefore have to be consistently applied when extending the CSE toolbox with models, processes, definitions or the like. CSE defines software development activities in a continuous and integrated fashion. For example, the Continuous * model [1] identifies continuous planning, continuous delivery, continuous security, continuous innovation and experimentation, continuous improvement, and other activities. The analysis of the CSE ecosystem surrounding the Continuous * model establishes three major themes: holistic approach, agile decision-making, and lean execution. These themes each contain a set of definitions,

principles, and guidelines that require a closer look. While similar in their goals, each theme has a different focal point for reflecting upon software engineering and how to better approach it.

Holistic Approach The history of software engineering has been characterized by counter-productive disconnects between important activities such as planning, analysis, design, implementation, and maintenance. Continuous software engineering (CSE) addresses this by taking a holistic view on the development of software systems: Processes across business, development, and operations are integrated to create a continuous flow of activities along with corresponding feedback cycles [9]. The Continuous * model by Fitzgerald and Stol [1] illustrates this: Development is no longer viewed as a stand-alone activity, but integrated with business and operations (see fig. 1.1).

Design Constraint 1 *The treatment design must holistically consider and involve all relevant processes across business, development, and operations.*

Agile Decision-Making Recognizing a need for flexibility and rapid adaptation, the "Agile Manifesto" signed by influential computer scientists outlines the following values for software development [19]:

- individuals and interactions (over processes and tools),
- working software (over comprehensive documentation),
- customer collaboration (over contract negotiation),
- responding to change (over following a plan).

The primary goal is to avoid unnecessary activities and focus on incremental as well as iterative development to increase productivity while being able to react to change at any time. This is expressed in process models like Extreme Programming (XP) or Scrum [see 6, 5].

Design Constraint 2 *The treatment design must enable agile decision-making, allowing projects to react to changing requirements, information, and circumstances.*

Lean Execution Lean development was adapted from the production management concept "lean manufacturing". While production and product development are not identical, the core principles have been successfully transferred to software development.

These principles aim to focus on value creation and flow, thereby reducing the time between an discovery of a need and delivery of value [7]:

- eliminate waste: avoid any activity or output that does not add value,
- amplify learning: treat development as exercise in discovery instead of production,

- decide as late as possible: keep options open, especially when faced with uncertainty,
- deliver as fast as possible: accelerate feedback and discovery cycles for learning,
- empower the team: improve technical and process decisions through self-organization,
- build integrity in: preserve cohesiveness and usefulness of software over time,
- see the whole: focus on overall performance to avoid suboptimization.

Design Constraint 3 *The treatment design must adhere to principles of lean development, establishing a continuous flow of value-creating activities.*

Resource Scarcity As described in section 1.1.2, constraints in decision-making can be understood as conditions of an optimization problem that the solution must satisfy. They can be "hard constraints" (mandatory satisfaction) or "soft constraints" (variable satisfaction).

It's not only that *decisions* that don't consider these constraints appropriately risk being wasteful or even exceeding the project's boundaries – any *activity* that is ignorant of project constraints will reduce the project's efficiency.

Design Constraint 4 *The treatment design must establish processes and systems that are efficient regarding time and resources of the project.*

Cognitive Biases As described in section 1.1.2, human decision making often shows systematic mistakes caused by shortcuts in the way decisions are made; these patterns of deviation from norm or rationality in judgment are called *cognitive biases.*

The following biases are relevant for treatment design and derive corresponding design constraints:

Design Constraint 5 *The treatment design must account for optimism bias and cannot assume realistic estimations of costs.*

Design Constraint 6 *The treatment design must account for confirmation bias and cannot assume immediate consideration of new facts.*

Design Constraint 7 *The treatment design must account for sunk cost fallacy and cannot assume that incorrect decisions will be reversed simply on a factual basis.*

Lean Principle	Description
Eliminate waste	Recognize activities or artifacts that are not adding value (e.g. partial work, waiting, defects) and reduce or remove them.
Amplify learning	Use fast feedback cycles to enable a continuous problem-solving process based on real insights and experimentation.
Decide late	Delay critical decisions and commitments until the can be based on facts to keep a capacity for change at all times.
Deliver fast	Develop in short cycles to generate value as quickly as possible and continuously collect feedback for learning.
Empower the team	Build projects around people and their skills, enable and motivate them, encourage progress, remove impediments.
Build integrity in	Make all parts of the product or system work together, both as perceived by the customer and between technical components.
Optimize the whole	Consider the entire system and prioritize holistic and long-term gains instead of just small, localized improvements.

Table 2.1: Seven principles for lean development (adapted from [7]).

Wicked Problems So-called *wicked problems* are defined as especially difficult or even impossible to solve due to special characteristics [57]. Rittel and Webber [66] initially defined the characteristics of wicked problems in the context of social policy planning. Conklin [67] later generalized them to suit other areas of problem solving and decision-making:

1. The problem is not understood until after the formulation of a solution.
2. Wicked problems have no stopping rule.
3. Solutions to wicked problems are not right or wrong.
4. Every wicked problem is essentially novel and unique.
5. Every solution to a wicked problem is a "one shot operation".
6. Wicked problems have no given alternative solutions.

While a truly wicked problem is characterized by all of these properties and not every problem in CSE will be wicked, this still poses an issue in problem solving: There might not be an optimal solution or, even if there is, it might not be possible to find it.

Design Constraint 8 *The treatment must account for the occurrence of "wicked problems" and cannot assume that a definite optimal solution can be found.*

2.1.2 Functional Requirements

From the description of the problem context and project environment, functional requirements for decision-making processes and decision support systems in the context of

continuous software engineering become apparent. Treatment design, both of processes and systems, must consider the main activities that guide decision-making within CSE. To elicit requirements, it analyzes necessary activities and desired outcomes in the problem context. The following requirements are derived from the problem context and defined so as to fulfill the desirable qualities of requirements.

Bruegge and Dutoit [36] and Zowghi and Gervasi [17] define the "4 Cs" for requirements that ensure that they are complete (all relevant requirements are captured), correct (requirements accurately reflect stakeholder needs), consist (requirements do not contradict each other), and clear (requirements can be understood easily and correctly). Bruegge and Dutoit [36] additionally asks that requirements are realistic (satisfiable under constraints of the target environment), verifiable (their satisfaction by the system can be tested), and traceable (recognizable throughout research activities).

Functional requirements for Continuous R&D are elicited from the CSE environment and aim to capture its decision-making processes. The following core activities of CSE form the loop across business, development, and operations and thereby guide the CSE organization. All are concerned with decision-making, either providing information required by decisions or utilizing information to make decisions. It is noteworthy that within CSE itself there is a balance between responding and planning as well as between exploration and refinement. Requirements for Continuous R&D therefore aim to capture these complementary aspects of CSE.

Continuous Innovation Innovation implies novelty but is not the same as invention. Instead, innovation refers to the process of creating or improving a product to create value. This can be based on new ideas, methods, or technologies, or on applying existing ones in a better way [10, 1, 16]. In the context of CSE, Continuous Innovation is defined as a "sustainable process that is responsive to evolving market conditions and based on appropriate metrics across the entire lifecycle of planning, development and run-time operations." [1] Lean differentiates two basic methods for improvement, applicable to both product and process: radical change ("Kaikaku") and incremental improvement ("Kaizen") [1, 33]. The focus is on enabling *disruptive* innovation ("Kaikaku") as opposed to step-wise evolution. The key aspects for decision making are responsiveness, decisions based on metrics, and sustainability. The learning aspect of lean development has furthermore been applied in startups and software product development, shortening time to market and involved risk [32, 33].

Functional Requirement 1 *The treatment must capture problems and solutions, both actively by exploring the problem and solution space through research activities and passively by receiving input from stakeholders.*

Functional Requirement 2 *The treatment must estimate the inherent variability of problems and solutions, the value of solving problems, the cost of implementing solutions, as well as the uncertainty inherent in these facts.*

Functional Requirement 3 *The treatment must discover and seize opportunities for problem-solving by matching low-cost solutions to high-value problems, regardless of when either becomes known to the organization.*

Continuous Planning Planning as the consideration of possible future outcomes of actions is traditionally applied like "predicting the future", i.e. the plan is then followed to the letter and irrevocable decisions are made early on [see 68]. In contrast, it can also be applied like a learning process if predictions are updated regularly and irrevocable decisions are made as late as possible [7]. Consequently, Fitzgerald and Stol [9] define Continuous Planning as "a holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution." An important contrast is planning ahead to be able to make reasonable decisions and manage risks, but responding to change as necessary, as opposed to "following a plan" [cf. 19]. As an analogy, Suchman [69] as well as Bruegge and Dutoit [36] describe the navigation strategy of Polynesian/Micronesian fishermen who navigate without maps and instruments. Instead, they rely on the position of stars for orientation and interpret changing environmental circumstances. These "situated actions" focus on local interactions with the environment instead of a preconceived plan. It is similarly important to differentiate planning on strategic, tactical, and operational level and appropriately select the level of detail as well as time frame [41]. The key aspect here is a decision-making process that reduces risk but keeps flexibility intact.

Functional Requirement 4 *The treatment must validate known problems and solutions as well as their connections using research activities and capture the updated uncertainty involved in these findings to achieve a level of confidence required for decision-making.*

Functional Requirement 5 *The treatment must support continuously identifying and prioritizing the need for further research based on estimated uncertainty inherent in all currently known facts.*

Functional Requirement 6 *The treatment must support continuously identifying and prioritizing potential implementation activities based on benefit-cost ratio of the involved problems and solutions.*

Continuous Experimentation An experiment in the scientific sense is a research procedure to test a hypothesis, that is to support, refute, or validate it [70, 71]. Continuous Experimentation in CSE is characterized as "based on experiments with stakeholders consisting of repeated Build-Measure-Learn cycles" [1]. Its goal is to use experiments to amplify learning, exploring a search space of unknown possibilities. The insights are useful for both large-scale innovation and small-scale improvement. It is important

to note that experimentation in this sense loosely means "testing an assumption" as well as "trying out something". The term experiment in this context is both restricted and softened: It is often used to refer to testing features in the production environment involving real users, but at the same time not necessarily implies the design of a controlled experiment but anything from a quasi-experiment to an observational study [see 31, 72, 30].

Functional Requirement 7 *The treatment must support experimentation, formulating assumptions that are critical to a project's success and testing them using research methods that are suitable for both exploring questions and testing hypotheses.*

Functional Requirement 8 *The treatment must support experimentation in both laboratory and real-world conditions by offering mechanisms that allow testing of an artifact before it is released as well as while it is being used in the production context.*

Continuous Improvement Continuous Improvement in CSE aims for incremental improvement ("Kaizen") instead of radical change ("Kaikaku") [1, 33]. and is "based on lean principles of data-driven decision-making and elimination of waste, which lead to small incremental quality improvements that can have dramatic benefits and are hard for competitors to emulate." [1] Decision-making should therefore be data-driven and make use of marginal gains, not only aim for big transformations.

Functional Requirement 9 *The treatment must guide how results from research activities and experiments are used in data-driven decision-making by defining suitable methods for collecting, verifying, and evaluating corresponding data.*

Functional Requirement 10 *The treatment must facilitate continuous learning from research activities and experiments to iteratively gain knowledge about the problem and solution space, increase confidence in facts and quality of decisions.*

2.1.3 Nonfunctional Requirements

Nonfunctional requirements describe aspects of the system which are not directly related to the functional behavior of the system [36]. The nonfunctional requirements for Continuous R&D are derived from the problem context and the CSE environment to capture additional goals for treatment design. They are held to the same quality criteria as functional requirements: completeness, consistency, clarity, correctness, realism, verifiability, and traceabilty. They are organized according to the FURPS+ model that categorizes software requirements into the categories functionality, usability, reliability, performance, and supportability with the latter four capturing so-called "quality requirements" and the "+" indicating additional constraints, also called "pseudo requirements" [73, 36].

Usability Users should be able to learn how to use a process or system, as well as how to provide inputs for it and utilize its outputs.

Nonfunctional Requirement 1 *The treatment should be easy to learn for all relevant organization members, provided that they are sufficiently familiar with the principles and techniques of CSE.*

Nonfunctional Requirement 2 *The treatment should facilitate collaboration and exchange of information between multiple roles within the CSE organization.*

Nonfunctional Requirement 3 *The treatment should be applicable to CSE projects regardless of their size, maturity, or other characteristics.*

Reliability Required functions should perform reliably both in terms of stable execution without errors and in terms of producing the same outputs for the same inputs under the same conditions.

Nonfunctional Requirement 4 *The treatment should consistently support decisions by providing repeatable and thereby verifiable ways to generate insights.*

Nonfunctional Requirement 5 *The treatment should prevent bad or premature decisions by making it obvious that their basis is insufficient.*

Performance Processes and systems have quantifiable attributes such as availability of a function, speed of reacting to input, throughput when performing a function, or accuracy of its result.

Nonfunctional Requirement 6 *The treatment should allow the CSE organization to react to new input at any time, regardless of the activities in progress.*

Nonfunctional Requirement 7 *The treatment should not enforce a linear sequence of activities but allow concurrent execution that is only governed by the organization's resources and priorities.*

Nonfunctional Requirement 8 *The treatment should allow the CSE organization to solve problems as well as required, but not necessarily optimally.*

Supportability Processes and systems have to change after their introduction, for example to adapt them to additional concepts or to fix defects in the existing implementation.

Nonfunctional Requirement 9 *The treatment should be adaptable to account for the individual differences or additional concepts in the context of an organization.*

Nonfunctional Requirement 10 *The treatment should be easy to maintain as new technologies or techniques emerge or deficiencies are discovered over time.*

Implementation Processes and systems may have explicit constraints regarding their implementation, for example requiring the use of specific tools and technologies to introduce or operate them.

Nonfunctional Requirement 11 *The treatment should not require specific tools, techniques, or technologies but rather be agnostic to the specifics of the organizational environment.*

Interface External processes and systems may add constraints for the integration of functionality or exchange of information.

Nonfunctional Requirement 12 *The treatment should be compatible with all commonly used tools, techniques, or technologies of the CSE ecosystem.*

Operation The administration and management of processes and systems may include additional constraints to the treatment design.

Nonfunctional Requirement 13 *The treatment should not require any additional roles to introduce, control, or maintain it, provided the organization already has adequate roles for process management and change management.*

2.2 Design of CRDM

Having sufficiently defined the problem context, design science conducts treatment design to come up with a suitable solution [63]. This is the next phase in the design cycle for CRDM and it investigates the knowledge question 2: Which concepts are central to scientific problem solving? This requires answering the following research questions:

1. Which roles and activities are involved in scientific problem solving?
2. Which concepts does this comprise and how do they relate?
3. Which interactions occur within scenarios and between entities?

In OOSE, the activity of analysis structures and formalizes the requirements specification to produce an Analysis Model [36], in this case the CRDM Analysis Model. The above research questions are answered by compiling a Functional Model, Object Model, and Dynamic Model, respectively. This formalization might lead to new insights, the discovery of errors in the requirements, or the need to go back and clarify or even redefine requirements. Section 2.2.1 composes the CRDM Functional Model which defines a taxonomy of actors involved in Continuous R&D. It then describes scenarios these actors perform or participate in as well as associations between scenarios. Section 2.2.2 describes the CRDM Object Model which captures the central elements from the requirements specification. It focuses on the relevant concepts involved in, interacting with, or manipulated by the system and describes entities, properties, and associations. Section 2.2.3 describes the CRDM Dynamic Model that captures system behavior by assigning responsibilities to entities and describing their interactions. Behavior can be described as a series of activities by actors, a sequence of interactions between entities, or state transitions in an entity. Figure 2.3 shows the exemplary parts of an analysis model to illustrate their content and relationship by using suitable UML diagram types.

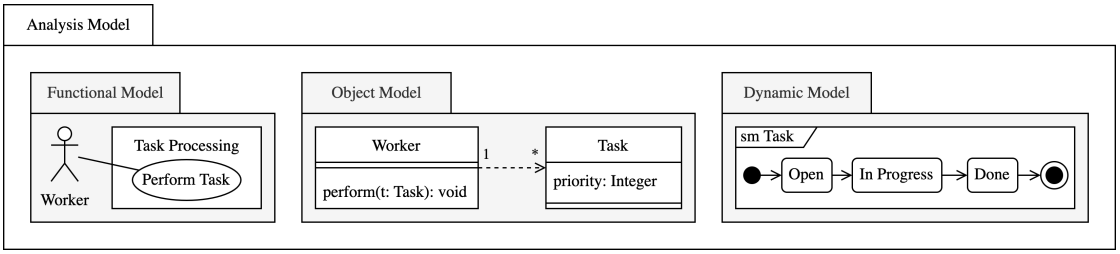


Figure 2.3: Example for the content of and relationship between Functional, Object, and Dynamic Model within the Analysis Model according to OOSE [36].

2.2.1 Functional Model

The foundation of CRDM is the description of Continuous R&D from the stakeholders' point of view, i.e. the functional model in OOSE [36]. This includes a description of the participants of Continuous R&D an overview of the use cases they participate in, and descriptions of each use case. The functional model is generated by analyzing the project environment as well as the functional and nonfunctional requirements described in the CRDM ecosystem. Figure 2.4 depicts CRDM's taxonomy of actors that centers around the CSE **Organization** and differentiates internal **Member**s from external **Stakeholder**s. The organization can have an arbitrary number of both, but members can only be associated with one organization while stakeholders could interact with other organizations as well. Stakeholders are differentiated into **Customer**s that provide problem statements for the organization to solve and commission the research and development effort to solve them. In addition to this, **Domain Expert**s possess expertise in the problem domain and provide additional insights, but are not necessarily buyers or users of the developed solution.[2] Within the organization, **R&D Manager**s lead the research and development effort, requiring both interacting with external stakeholders and coordinating internal efforts. R&D managers therefore are the *key decision makers* in Continuous R&D. They collaborate closely with **Analyst**s who focus on decision support and therefore collect and evaluate information from internal and external sources, both through actively research efforts and passively through stakeholder interaction. **Developer**s are responsible for implementing the solutions decided upon by the organization and simultaneously serve as experts for the solution domain, providing respective input for decision support.

Table 2.2 summarizes these roles defined by CRDM. It is noteworthy that this taxonomy is extensible both regarding internal and external actors. For example, a real-world software organization may include a greater variety of more specialized roles such as program managers, project managers, product managers, testers, security engineers, infrastructure engineers, data engineers, data scientists, business analysts, and others. Likewise, external stakeholders may be differentiated into end users, buyers, reviewers,

[2] It is critical for the CSE organization to build up knowledge in both problem and solution domain [25]. CRDM defines domain experts as stakeholders to emphasize that they are not necessarily members.

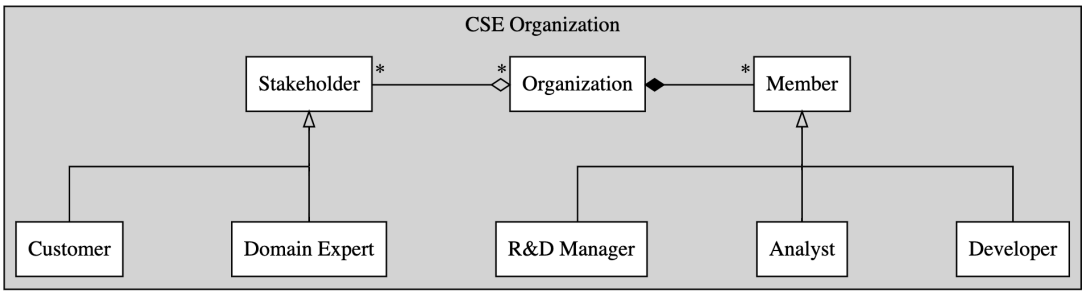


Figure 2.4: Taxonomy of actors in CSE, differentiating between the organization's members and its stakeholders; the model is meant to be extensible on both sides.

regulators, and others. CRDM focuses on the fundamental roles within Continuous R&D and abstracts them into the identified actors. Similarly, it does not differentiate between human and machine actors and would allow a domain expert or analyst role to be filled by a human, an information system, or both.

Figure 2.5 depicts the functional model of CRDM that describes the use cases performed by the roles described above. It highlights three root use cases in which different roles collaborate and which define the main areas of Continuous R&D. **Solve Problem** represents the root use case of the problem solving process and involves the R&D manager and the customer, optionally additional domain experts for support. It consists of capturing and matching problems and solutions, managing the execution of the R&D process to deliver the solutions, and capturing feedback to generate further insights into the problem and solution domain. **Support Decision** represents the root use case of the decision support process and involves the R&D manager along with analysts. It involves assessing the uncertainty in the problem and solution domain, identifying need for research to reduce it, and finding opportunities for implementation. The insights generated by these use cases are utilized in the problem solving process. **Execute Task** represents the root use case of the continuous execution process and involves analysts and developers. It covers both research activities to support decisions and implementation activities to execute decisions. Research includes finding or prioritizing problems, finding or estimating solutions, or evaluating the suitability of solutions for problems. Implementation includes any activity necessary to prepare, develop, verify, deploy, and validate a solution. Execution is asynchronous and there might be an arbitrary number of independent R&D activities at any point in time.

Since CRDM models Continuous R&D on a fundamental level, it describes abstract use cases involving generic roles and entity classes instead of concrete scenarios involving specific actors and objects [36, see]. The following sections describe use cases grouped according to the associations described above. Section 2.2.1.1 describes problem solving use cases, and section 2.2.1.2 describes decision support use cases, and section 2.2.1.3 describes continuous execution use cases. Appendix B.1.1 contains detailed tables of these use cases with participating roles, preconditions, flow of events, and postconditions. The CRDM functional model omits how and by whom use cases are initiated since, due to the dynamic nature of CSE, both other use cases and external events could act as triggers. Therefore, the dynamic model in section 2.2.3 provides structure for how use cases relate to each other.

Role	Description
Customer	Describes problem statements and commissions R&D efforts.
Domain Expert	Provides additional knowledge about the problem domain.
R&D Manager	Coordinates R&D efforts by making key decisions.
Analyst	Collects and evaluates information for decision support.
Developer	Implements solutions and acts as solution domain expert.

Table 2.2: Identifiers and descriptions of roles defined by CRDM.

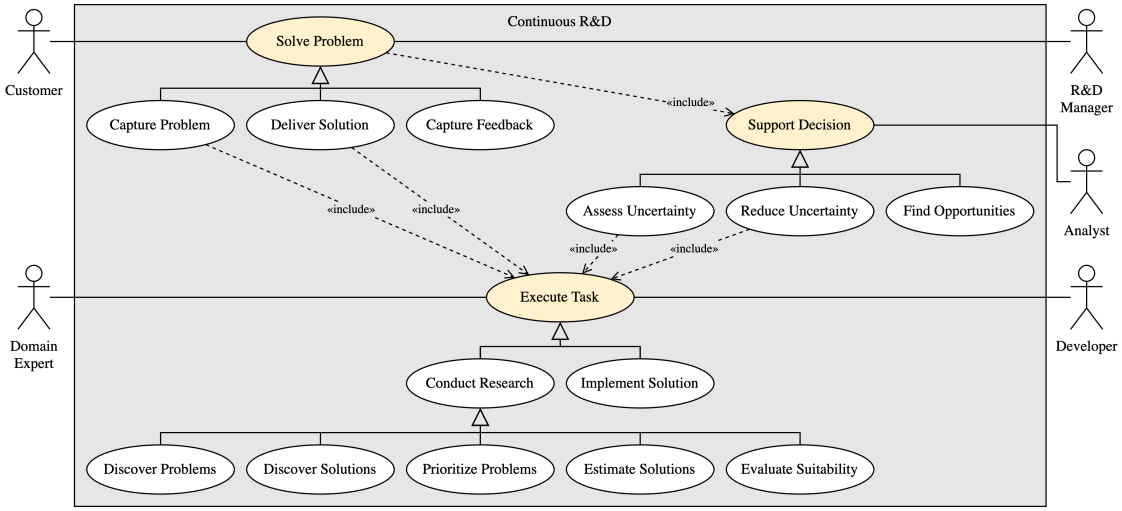


Figure 2.5: CRDM functional model, key use cases of Continuous R&D highlighted, associated with *primary* participating roles (all use cases may involve collaboration of all roles).

2.2.1.1 Problem Solving

In problem solving, the R&D manager collaborates with the customer and domain experts on capturing the relevant problems in the domain, delivering suitable solutions and subsequently capturing feedback to further improve and extend the delivered solutions. In doing so, the R&D manager is in charge of decision making and overseeing the execution while the customer provides input and feedback regarding the content and priority of problems as well as the suitability of solutions. The domain experts support this process with expertise on the details of problem and solution domain, enabling the Continuous R&D organization to make the right decisions and deliver the right solutions.

Use Case 1 (Capture Problem) *R&D manager captures a problem described by the customer with additional input from domain experts.*

Problems in the knowledge base represent the needs of customers to be fulfilled by the organization. They need to be captured with all information necessary to subsequently make prioritization and implementation decisions. The R&D manager is responsible for capturing them appropriately in the knowledge base, using the support of domain experts to correctly specify details of the problem, connections to other problems, the value of solving the problem, and potential solutions. Table B.1 provides a detailed description of the use case.

Use Case 2 (Deliver Solution) *R&D manager guides the delivery of a solution to the customer to solve a previously captured problem.*

The core activity of the R&D manager is to guide the research and development activities of the Continuous R&D organization with the goal of delivering solutions to the

customer. To optimize the cost/benefit ratio, the R&D manager chooses solutions that solve valuable problems with reasonable effort. This involves analyzing both problem and solution space in terms of predicted solution benefit, available solution alternatives and associated cost factors. It also requires taking into account the type and degree of uncertainty associated with each information in the knowledge base. If a decision involves too much risk according to the parameters of the organization, the R&D manager may postpone it. Upon deciding which solutions to work on, the R&D manager commissions implementation tasks that describe the desired outcome of the implementation activity as well as it priority. Once implementation is finished, the R&D manager oversees delivery of the solution to the customer. Table B.2 provides a detailed description of the use case.

Use Case 3 (Capture Feedback) *R&D manager captures feedback by the customer and domain experts on knowledge base contents or implemented solutions.*

To enable iterative implementation, learning, improvement, feedback loops are established to collect insights about contents of the knowledge base (such as correctness of captured problems, completeness of considered solutions, realism of estimations) as well as implemented solutions (concerning aspects such as correctness or effectiveness, newly discovered follow-up problems). Feedback can be provided actively (e.g., by the customer and domain experts) or passively through monitoring, measuring, and analytics methods. Thus, feedback can be based on quantitative data, their expertise or subjective experience. Feedback can be collected at different points in time: early analysis and prototyping, continuous involvement during development, reviews before delivering the solution, usage after delivering the solution. The R&D manager captures feedback in the knowledge base, updating existing entities and links or adding new ones. Table B.3 provides a detailed description of the use case.

2.2.1.2 Decision Support

In decision support, the R&D manager collaborates with analysts on assessing the uncertainty within problem and solution domain, reducing it through respective research activities, and finding opportunities for delivering suitable solutions to problems that are relevant to the customer. In doing so, the R&D manager is responsible for actually making the decisions under uncertainty, e.g., which problems to work on and which solutions to apply to them. The analysts are in charge of capturing the details necessary to assess the uncertainty, guiding corresponding research activities, and using generated insights to update the knowledge base with information that enables the Continuous R&D organization to discover opportunities for delivering value to the customer.

Use Case 4 (Assess Uncertainty) *Analysts support R&D manager in assessing the locations, types, and degrees of uncertainty in the knowledge base.*

In order to manage risks in decision-making, the R&D manager needs to be aware of uncertainty involved in both problem and solution domain. To support this, analysts help with assessing the contents of the knowledge base with regards to where uncertainty is

inherent, which type(s) of uncertainty play a role, and to which degree. Information about relevant locations, types, and degrees of uncertainty is captured in the knowledge base, attached to the affected entities. For example, the cost of a solution might be slightly uncertain due to rapid technological advancement, its suitability for a problem might be highly uncertain due to high variability in the problem space, and the benefit of solving the problem might be moderately uncertain due to missing information. Uncertainty is used to express how confident the Continuous R&D organization is to make decisions based on the affected information, e.g., on a scale from 0 % to 100 %. Naturally, these factors and their impact on the project can change over time, so uncertainty needs to be updated along with any other information in the knowledge base. Table B.4 provides a detailed description of the use case.

Use Case 5 (Reduce Uncertainty) *Analysts reduce uncertainty in the knowledge base using research activities and guidance from R&D manager.*

Having assessed the location and level of uncertainty in the knowledge base, the R&D manager reviews the potential risk involved for upcoming decisions. The R&D manager identifies areas where the risk/reward ratio of deciding based on existing facts and their inherent uncertainty is not reconcilable with the organization's risk management. In these situations the uncertainty needs to be reduced before making a decision. The R&D manager therefore may commission research activities to reduce uncertainty before making decisions. Analysts then identify the type and source of each factor of uncertainty in the affected area of the knowledge base. They select a research strategy and methodology suitable to the type of uncertainty, the desired result (i.e., how much the uncertainty needs to be reduced), and the permissible time and effort to be spent on the research activity. A corresponding research task is created in the organization's task backlog, prioritized based on the importance and urgency of the research, and executed according to the organization's execution process model. After the research task has been performed, analysts evaluate the results and update the knowledge base accordingly. The R&D manager can then use the updated information to re-assess the situation and, hopefully, make a better decision than previously possible. Table B.5 provides a detailed description of the use case.

Use Case 6 (Find Opportunities) *R&D manager and analysts use insights from knowledge base to discover opportunities for problem solving.*

In addition to the demand-driven character of the problem solving process described in section 2.2.1.1, CRDM allows for opportunistic behavior of CSE organizations. Opportunism in the context of Continuous R&D means using input from stakeholders as well as insights from both research and development to recognize and exploit chances for problem solving with a good cost/benefit ratio. This cost/benefit ratio can stem from the cost of solutions decreasing through technological advancements, the benefit of solving a problem increasing due to changes in the market, or previously unknown solutions or problems being captured that fit with existing elements of the knowledge base. The R&D manager is responsible for monitoring the knowledge base regarding changes

that could imply such opportunities. When an opportunity is recognized, analysts may additionally validate the underlying facts to ensure a reliable basis for decision-making. If the cost/benefit ratio as well as the risk/reward ratio of the opportunity match the organization's criteria, the R&D manager commissions an implementation task describing the required implementation activity. The implementation task is prioritized based on the importance and urgency of seizing the opportunity and then executed according to the organization's execution process model. Table B.6 provides a detailed description of the use case.

2.2.1.3 Continuous Execution

In continuous execution, the analysts and developers collaborate with domain experts on conducting the research activities necessary for decision support as well as the implementation activities necessary for problem solving. In doing so, the analysts are in charge of guiding the research activities, which are supported by domain experts to provide input on research design as well as input as part of the research. Developers support the execution of research and are in charge of the implementation activities, where the roles are reversed and analysts as well as domain experts support them with insights and feedback.

Research vs. Implementation CRDM differentiates between research and implementation as the purpose of a CSE organization's activities. This emphasizes whether the activity is meant to generate new insights to support future decisions or deliver solutions by executing on past decisions. In practice, however, the line between research and implementation might be blurred with continuous experimentation using the production environment to simultaneously advance the product and capture new information about both problem and solution domain [25, 30, 72]. CRDM therefore describes the intention behind an activity but then treats all of them identically as tasks in a queue to be executed by the organization's process model of choice [74].

Use Case 7 (Discover Problems) *Analysts and domain experts identify and capture problems in the problem space to be solved by the organization.*

Problem discovery can be prompted actively by the organization if they have or anticipate capacity to work on more problems that are currently captured in the knowledge base. Alternatively, it can be triggered by stakeholders reporting problems that they deem important for the organization to work on. In any case, analysts are responsible for investigating the problem and describing its implications for stakeholders. The problem is also broken down if appropriate, for example if it seems too complicated to be solved at once or if it contains parts that must be approached with different solutions. This process is supported by domain experts to make their knowledge of the problem space available for subsequent analyses and decisions within the organization. For this purpose, analysts capture the discovered and analyzed problems as entities in the knowledge base. Any information that is available at this point is captured as well, such as initial guesses

at problem benefit, connections to potential solutions, and type and degree of uncertainty associated with each piece of information. Table B.7 provides a detailed description of the use case.

Use Case 8 (Discover Solutions) *Analysts and domain experts identify and capture solutions in the solution space to be utilized by the organization.*

Similar to problem discovery, analysts, domain experts, and developers collect knowledge about available solutions by either receiving input from organization members and stakeholders or actively eliciting potential solutions from the solution domain. Developers are important participants of any solution-related use case due to their presumed expertise in the solution domain. Upon discovering solutions they capture available information about the solution, estimated costs, connections to known problems, and inherent uncertainty in the knowledge base. It's important to note that solution discovery is independent from problem discovery. Technologies in general or solution mechanisms in particular can be suggested to the organization or investigated by the organization without knowing yet which problems to apply them to. This allows the organization to collect and refine knowledge about the problem and solution domain in parallel while continuously watching for opportunities in problem solving. CRDM thereby allows non-linear problem solving, in contrast to linear approaches that first require compilation of a problem statement that is subsequently analyzed to design a solution. Table B.8 provides a detailed description of the use case.

Use Case 9 (Prioritize Problems) *Analysts and domain experts investigate problems and prioritize them according to their benefit when solved.*

In order to decide which problems to work on, the organization weighs risks and rewards with the reward side being determined by the benefit of solving each (partial) problem. Analysts and domain experts therefore investigate problems to estimate their benefit. The estimated benefit value and the inherent uncertainty are stored in the knowledge base. The organization then choses estimation and prioritization mechanisms that reflects their needs and constraints. Estimation could include the monetary value of selling the solution to a problem, the business value constrained by the problem, the strategic value of owning the intellectual property to its solution, the ethical value of improving on a situation, as well as any other factor that the organization and its stakeholders consider a benefit. Prioritization could use benefit values in isolation to rank problems or combine them with cost values of connected solution. It can (and should) also consider the confidence in benefit values, cost values, and suitability of the solution to provide a realistic metric for decision making. It's important to note that prioritization occurs not only once but as often as needed, such as whenever a new entity is added or the knowledge base is updated from research activities that are conducted to counteract uncertainty. Table B.9 provides a detailed description of the use case.

Use Case 10 (Estimate Solutions) *Analysts and domain experts investigate problems and estimate them according to their cost when implemented.*

In order to decide which solutions to employ, the organization weighs risks and rewards with the risk side being determined by the costs of implementing each (partial) solution. Analysts, domain experts, and developers therefore investigate solutions to estimate their costs. The estimated cost value and the inherent uncertainty are stored in the knowledge base. The organization then choses estimation mechanisms that reflects their needs and constraints. Estimation could include the monetary cost of purchasing or licensing a solution, the time and effort required to implement or introduce it, the complexity of the involved systems and processes, as well as any other factor that the organization and its stakeholders consider a cost. It can (and should) also consider the confidence in each of these factors to provide a realistic metric for decision making. It's important to note that estimation occurs not only once but as often as needed, such as whenever a new entity is added or the knowledge base is updated from research activities that are conducted to counteract uncertainty. CRDM makes no assumptions about the estimation and prioritization mechanisms employed as there is relevant literature on fundamentals as well as concrete techniques such as Berander and Andrews [75] and Trieflinger et al. [76] that address these issues. Table B.10 provides a detailed description of the use case.

Use Case 11 (Evaluate Suitability) *Analysts and domain experts investigate the connections between problems and solutions to evaluate their respective suitability.*

CRDM considers both the benefit of solving a problem and the cost of implementing a solution. However, in order to realize the implied cost/benefit ratio of a combination, the connection between each concrete problem and solution must be considered as well. Such a connection is an additional entity associated with a suitability value and its own factors of uncertainty. This means even if both a problem and a solution have been evaluated properly, the suitability of the particular solution to the particular problem might be unknown, uncertain, subject to change, depending on other factors outside the control of the organization, and so on. Analysts, domain experts, and developers therefore analyze and capture information about problem/solution connections, their suitability value, and inherent uncertainty in a similar fashion to the analysis and capture of the problems and solutions themselves. As mentioned above (use case 9), the resulting confidence in problem/solution connections should be integrated into the problem prioritization mechanism. Table B.11 provides a detailed description of the use case.

Use Case 12 (Implement Solution) *Developers execute implementation activities defined in the decision-making process to deliver selected solutions to selected problems.*

The results of research activities are used in decision support (use case 6) to continuously find opportunities for solving a valuable problem with an efficient solution under acceptable risk. When an opportunity is discovered, R&D manager, analysts, and developers define implementation activities required to actually deliver a solution as

part of problem solving. CRDM does not prescribe what needs to be done as part of an implementation task. In the context of continuous software engineering, it most likely includes development work but also might entail anything from design and conceptualization to delivery and communication. In the context of agile methodology, the solution most likely isn't finished with a single implementation activity but realized over time through incremental and iterative work. Implementation can require one or multiple implementation tasks that might be executed in parallel or serially and synchronized in time or asynchronously, depending on the problem and solution at hand as well as the organization's execution process model. Implementation activities might also assist research activities if mechanisms need to be established to conduct experiments in a lab or production environment. Developers execute the implementation activity as required and notify about the results within the organization. Delivery and communication with stakeholders is part of the higher-level problem solving process (use cases 2 and 3). Depending on the organization's delivery model, results may be delivered to customers in a controlled fashion or in a continuous stream of updates. Table B.12 provides a detailed description of the use case.

2.2.2 Object Model

According to Bruegge and Dutoit [36], an object model "describes the structure of a system in terms of objects, attributes, associations, and operations". In OOSE, the object model constructed during requirements elicitation and analysis is called analysis object model. Its purpose is to describe capture concepts from the application domain, in this case Continuous R&D. This approach is rooted in the Object Modeling Technique (OMT) described by Rumbaugh et al. [77]. Figure 2.6 provides an overview of the CRDM object model by defining Continuous R&D as a package that the CSE organization interacts with as well as the main packages within it.

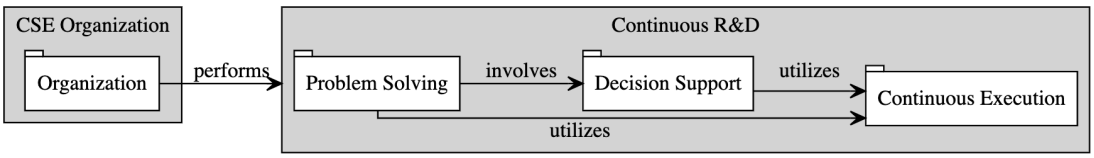


Figure 2.6: Overview of CRDM object model showing packages within Continuous R&D.

Within Continuous R&D, **Problem Solving** represents the overarching process which involves **Decision Support** for the stakeholders. Both in turn utilize **Continuous Execution** to gather insights for decision-making and to execute on decisions. Thus, Continuous Execution in the CRDM model covers both Continuous Development and Continuous Experimentation in the Continuous * model. A further look into the CRDM packages reveals the rationale for this design. Figure 2.7 shows the essential entities within the packages as well as their relationships.

The **Stakeholder**s of the CSE organization describe **Problem**s and the **Member**s of the organization work on the **Solution**s to these problems by following the **Process Model** of the continuous execution package. The process model processes **Task**s, performed by the organization members, particularly **Implementation Task**s that work on a solution. Executing tasks requires making **Decision**s which in turn require **Knowledge** that consists of **Fact**s. Gathering these facts and addressing the **Uncertainty** carried by them also requires work by the organization members. Therefore, tasks can also be **Research Task**s that generate insights: They address uncertainty by generating or refining facts, namely all elements of the problem solving package.

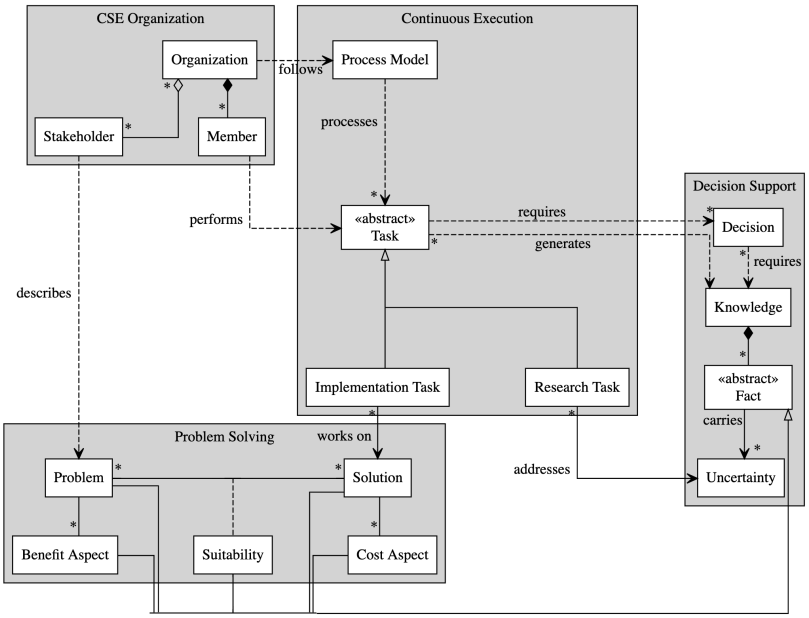


Figure 2.7: Essential packages and their associations within the CRDM object model.

2.2.2.1 Problem Solving

Continuous R&D considers problem solving as an open-ended process, repeating both endlessly over time as well as fractally on ever deeper levels. This approach fits into "continuous planning" in the Continuous * model for CSE [1] where plans are dynamic artifacts that evolve over time. The fractal structure is based on problem solving approaches such as IBIS [60] or CONSUL [78] that formulate needs as "issues" and approaches as "positions". In these models, entities can be decomposed into smaller parts for further analysis and resolution, enabling a "divide and conquer" approach. Figure 2.8 depicts the CRDM problem solving model. The stakeholders of the CSE organization describe **Problem**s in the **Problem Domain** (also called problem space or business domain). These problems can be approached with **Solution**s from the **Solution Domain** (also called solution space or application domain). Both problems and solutions can either be atomic elements to be acted on or decomposed for further analysis.[3]

The terms "problems" and "solutions" are used neutrally for issues and ways to address them [80], further described by Brown et al. [81] as depicted in fig. 2.9: Problems constitute a force within a context, e.g., a desire of a stakeholder that needs to be fulfilled or a barrier that keeps them from achieving a goal. A (beneficial) pattern exists where a solution solves a problem, generating a benefit for the stakeholder but also incurring consequences. An antipattern emerges when a solution becomes problematic, no longer solves the problem or exhibits mostly negative consequences. A refactored solution can rectify this, but in turn might generate follow-on problems that require entirely new

[3]This is analogous to the *composite pattern* in software design [79].

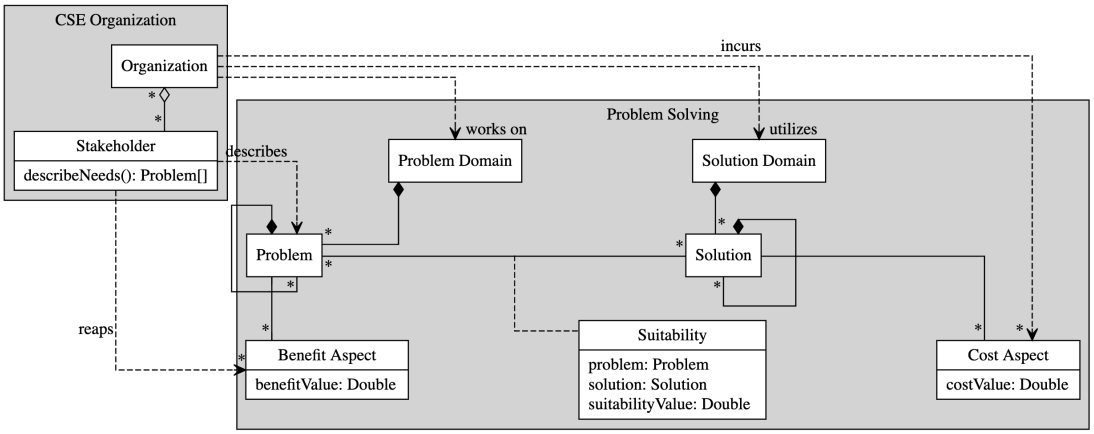


Figure 2.8: Elements of problem solving in Continuous R&D with links between problem and solution space.

solutions. CRDM incorporates this rationale, but distributes the forces, benefits, and consequences differently to better suit the decision-making process. From stakeholders' perspective, problems are associated with **Benefit Aspect**s that they would like to reap. Solutions, on the other hand, carry **Cost Aspect**s that the organization has to incur when implementing the solution. This implies a cost/benefit ratio between a specific pairing of problem and solution which will serve as the guideline for selection and prioritization of the organization's work. This value-driven approach has proven useful in prioritizing features [82, 34, 28]. The cost/benefit ratio is limited by the **Suitability**: How well a solution solves a problem and therefore how much of the problem's benefit can be reaped by incurring the solution's cost. However, decision-makers also need to factor in other criteria throughout the software life cycle such as project constraints, design goals, and nonfunctional requirements [83, 84].

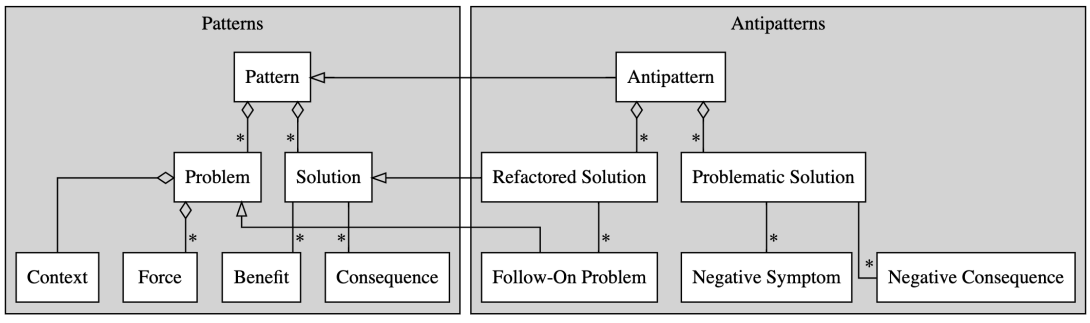


Figure 2.9: Visualization of the connection between problems and solutions as part of patterns and antipatterns, based on [81].

2.2.2.2 Decision Support

To allow dealing with a high degree of uncertainty in the problem or solution domain, CRDM views **Decision**s through the lens of decision support, as depicted in fig. 2.10. Decision support focuses on finding a valid and satisfactory solution, not necessarily the optimal one [41]. However, a decision still implies solving **Decision Problem**, even if the goal is not to find the optimal solution. Therefore, decisions are the result of weighing **Argument**s, considering **Feedback**, following a **Strategy**, and (most importantly in Continuous R&D) applying existing **Knowledge**. These elements form the **Rationale** that both guides the decision upfront and explains it afterwards. Knowledge embodies all **Fact**s that are acquired during the organization's activities. In the context of Continuous R&D, facts have to be assumed to carry **Uncertainty** that makes it difficult to assess the outcome of a decision.

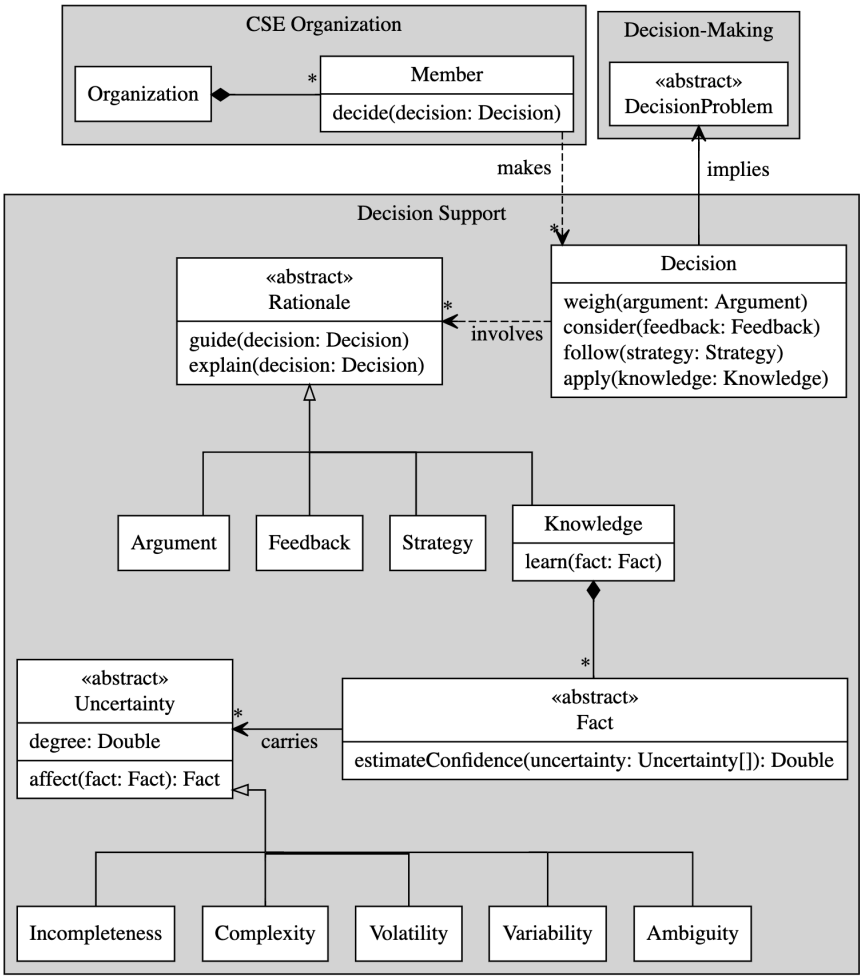


Figure 2.10: Elements of decision support in Continuous R&D describing decisions based on rationale, including knowledge affected by uncertainty.

Uncertainty Literature on problem solving and decision support identifies multiple sources for uncertainty and therefore obstacles for Continuous R&D. The type of uncertainty and the degree to which it affect a fact can be used to determine whether uncertainty needs to be reduced before a decision can be made or whether a calculated risk can be taken despite remaining uncertainty [85, 42, 43]. **Incompleteness** is the lack of availability of information, i.e., missing information or fragmentary information, making it harder to solve related problems [38]. Incomplete information should be differentiated from ambiguous information, meaning a lack of clarity that allows for multiple interpretations of the same piece of information [36, 37]. **Ambiguity** allows several plausible interpretations at once [86]. It therefore increases the difficulty of interpreting available information, i.e. the uncertainty that a particular interpretation is correct. Ambiguity is often associated with vagueness (an insufficiency to allow proper interpretation) but while ambiguity might be caused by vagueness, even clear information might be ambiguous [41, 86]. Both incompleteness and ambiguity necessitate research in order to alleviate the lacking quantity or quality of information. **Variability** refers to the lack of consistency in an area due to diversity, dispersion, or dependencies of its elements. It makes decisions among those elements harder since testing out all possibilities empirically might not be economical or feasible[4]. This necessitates continuous exploration of problem and solution space as well as a satisficing approach instead of optimization [38].

Volatility describes the chance of a situation changing as well as the nature or rate of this change. As a result of change, information might become incorrect, unavailable, or ambiguous, which induces uncertainty [87, 75]. This necessitates feedback cycles to detect change as well as iterating faster than the rate of change in order to stay reactive. Ideally, decision makers are able to anticipate change instead of incurring the delay of detection and the cost of adaption in retrospect. **Complexity**, in the sense of systemic complexity[5], is the degree to which a system's behavior is difficult to model due to characteristics like nonlinearity, emergence, spontaneous order, adaption, and feedback loops [39]. This can manifest in varying degrees which requires differentiation: While "complicated" means difficult to understand but ultimately knowable, "complex" indicates uncertainty that the model of system behavior is correct, and "chaotic" denotes a total lack of transparency or predictability [88]. Depending on the degree of complexity, a decision problem might be solvable with sufficient information, but it might still be hard or even impossible to predict the outcome of each possible decision [89, 90]. This necessitates reduction in complexity through scientific research as well as decomposition of the problem and solution space.

Decision Problems CRDM incorporates the characterizations of decision problems in requirements engineering [41] and rationale management [80], as illustrated in fig. 2.11: A decision problem fundamentally is described by its **Alternative**s to be decided on and the **Criteria** to evaluate the alternatives against. The mode of decision and its outcome can be distinguished into **Triage** (classifying each alternative), **Ranking**

[4] A high growth of options due to variability is named "combinatorial explosion" in mathematics.
[5] Not be confused with computational complexity, the difficulty to solve a problem algorithmically.

(ordering all alternatives), and **Selection** (selecting a subset of alternatives). Depending on the **Perspective** of the decision-maker, they will either require more requirement-centric decision support, focusing on the artifacts in question, or activity-centric decision support, guiding the analysis activities. The perspective is influenced by **Context** such as the type of system under construction, maturity of the organization, experience of the decision maker, availability of information, and similar factors. One important distinction based on the perspective is which **Category** the decision problem is treated as, ranging from structured (known problem, clearly defined process) to unstructured (novel problem, ambiguous process) with varying degrees of semi-structured (known problem, but ambiguous process) inbetween. When solving the decision problem with the goal of satisficing, its **Level** must be taken into account to consider both the **Impact** and **Time Horizon** of the resulting decision. Decisions on the **Strategic Level** typically carry a large-scale impact and have a long-term time horizon, thereby defining **Strategic Objective**s. These are to be achieved on the **Tactical Level** with a medium impact and time horizon, defining the **Tactical Planning**. The plan is to be fulfilled on the **Operational Level**, defining tasks that in turn will require the next set of decisions.

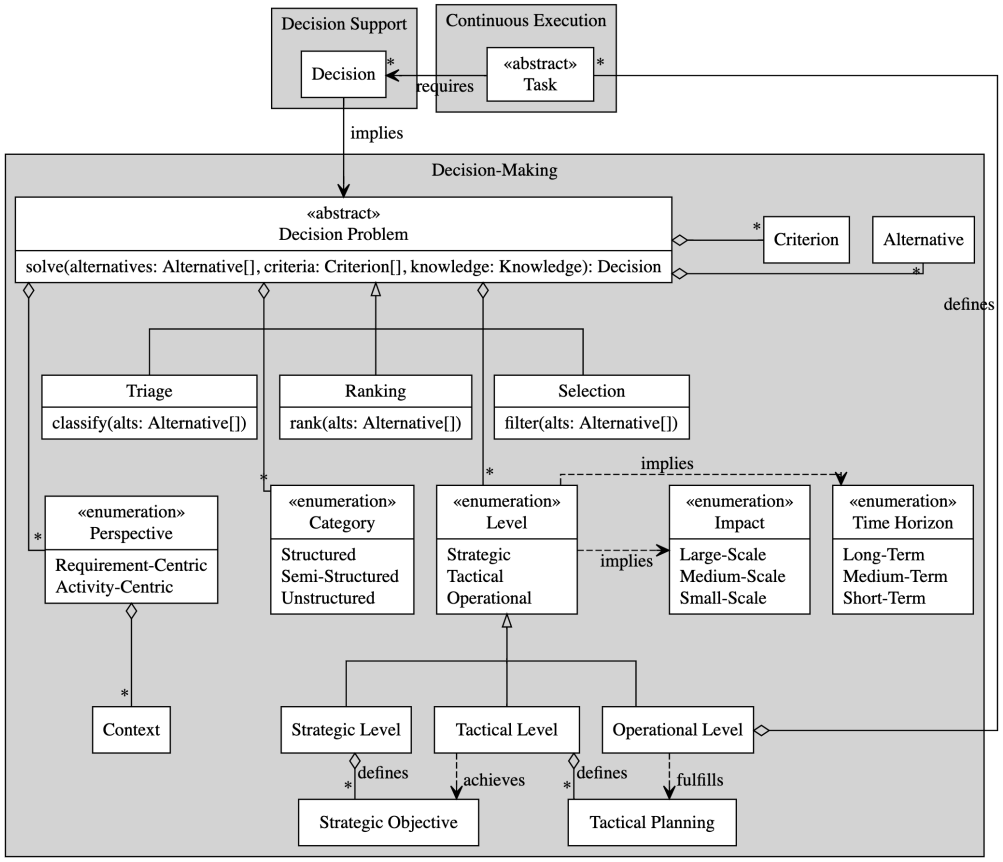


Figure 2.11: Model of decision-making as the process of solving decision problems of different types and on different levels, based on [41, 80].

Knowledge Management Due to uncertainty in problem and solution domain, the input for decision-making is critical for avoiding errors and thereby expensive and time-consuming correction cycles [91]. This is the objective of knowledge management: Models for decision support, decision-making, and rationale management structure the information to be used in preparing, making, and documenting decisions [92]. However, since both the amount and interdependence of information can grow large for non-trivial projects in high-uncertainty domains, the knowledge base needs to be managed correctly for effective decision support [93, 85, 84]. CRDM is concerned with the elements in the knowledge base, the points of variation, the degree of variation at each point, and the dependencies between them. The QOC model for design rationale structures the space of decision problems ("design space") into questions, options, and criteria [94]. These elements are reflected by decision problems, alternatives, and criteria in CRDM. The QUARC model for knowledge management generalizes from both QOC and IBIS by defining an abstract entity, the quarc, to represent every concrete item in the knowledge base [95].. In CRDM, this abstract construct is the fact which captures entities and relationships alike. The Orthogonal Variability Model (OVM) for variability management defines a meta-model for variability in conjunction with arbitrary development artifacts [96]. OVM has been integrated with issue-based approaches to rationale management [97, 98], suggesting its compatibility with CRDM.

Figure 2.12 depicts how CRDM incorporates the main elements from QOC, QUARC, and OVM into decision-making and decision support. As part of a **Decision Problem**, each **Alternative** covers certain **Variation Point**s in the problem and solution domain and chooses a specific constellation among all **Variant**s associated with them. As defined in the core object model in the beginning of this section, all parts of the problem solving model manifest as facts in the knowledge base. Thereby, **Fact**s represent variation points relevant for decision-making and realize specific variants of that variation point. For example, there might be multiple cases of a problem and multiple technologies usable for a solution. This relationship is characterized by **Variability Dependencies** between variation points and variants, i.e., a variation point can make it optional or mandatory to choose either one or multiple of the variants. For example, there might be multiple cases of a problem and reaping its benefit requires solving at least one of them, but solving more cases might be even more beneficial. Additionally, there can be **Constraint Dependencies** both within and inbetween variants and variation points. For example, solving one problem might require solving other problems first or a chosen technology might restrict future choices. Appendix B.1.2 contains further definitions from OVM regarding variability management.

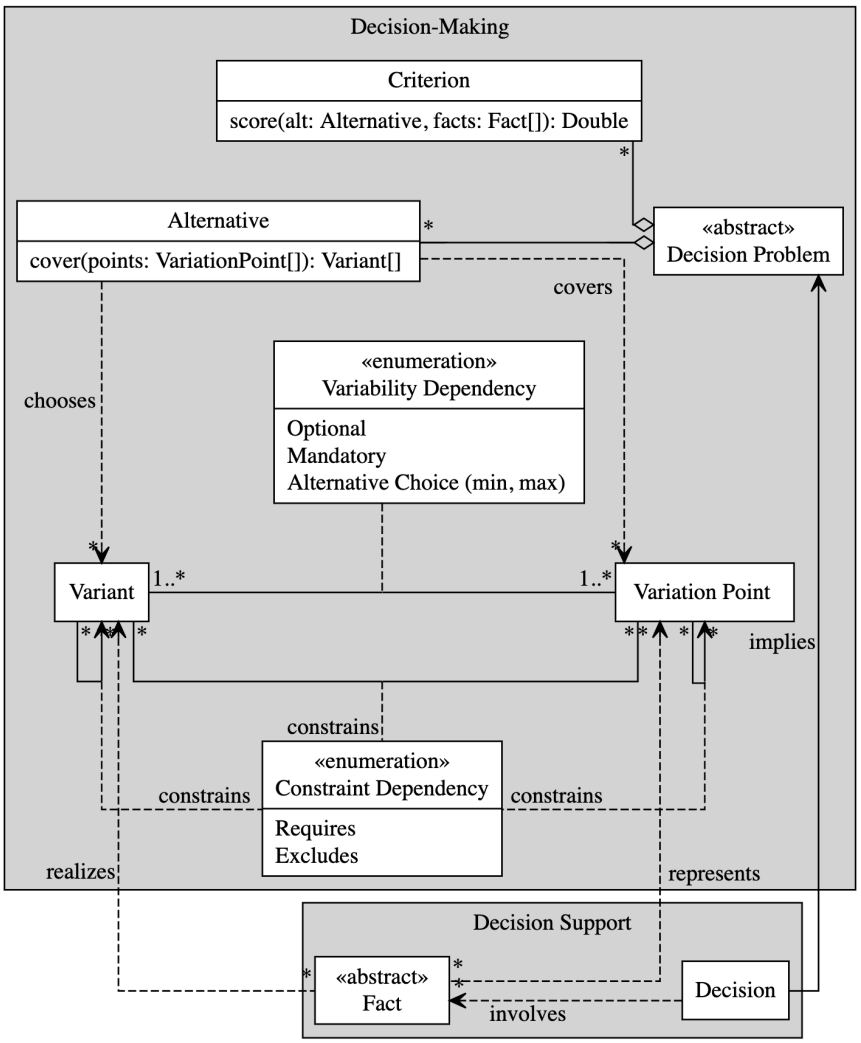


Figure 2.12: Knowledge management connecting decision problems with alternatives and criteria to facts, variability, and constraints, based on QOC [94], QUARC [95], and OVM [96].

2.2.2.3 Continuous Execution

The higher the degree of uncertainty in a domain, the more the decision-making needs to be driven by continuous analysis and experimentation [88]. Fitzgerald and Stol [1] argue that CSE is not only about development and deployment techniques but a "holistic view of a software production entity". CRDM adheres to this desired integration between planning, development, and feedback. It defines a common execution model for activities that the Continuous * model for CSE describes as continuous delivery, continuous experimentation, and continuous innovation [1]. It does so by adapting the CSEPM meta-model for continuous processes by Krusche and Bruegge [74], as depicted in fig. 2.13: Every activity in Continuous R&D is treated as a **Task** to be carried out by a suitable member of the CSE organization. Tasks are part of a **Task Queue** and when executed can generate follow-up tasks which then join the queue. The task queue is processed by **Process Activities**, taking either one or multiple tasks at once. Process activities are triggered by **Process Event**s, such as the beginning of an iteration or the completion of a task or input from a stakeholder. This event-driven model allows the Continuous R&D process to be flexible and continuously react to change in a nonlinear fashion. Both process events and process activities are defined by the **Process Model** the organization is following and can therefore be tailored to its environment, requirements, and constraints. CRDM extends this model with specific task types to cover activities across continuous delivery, experimentation, and innovation.

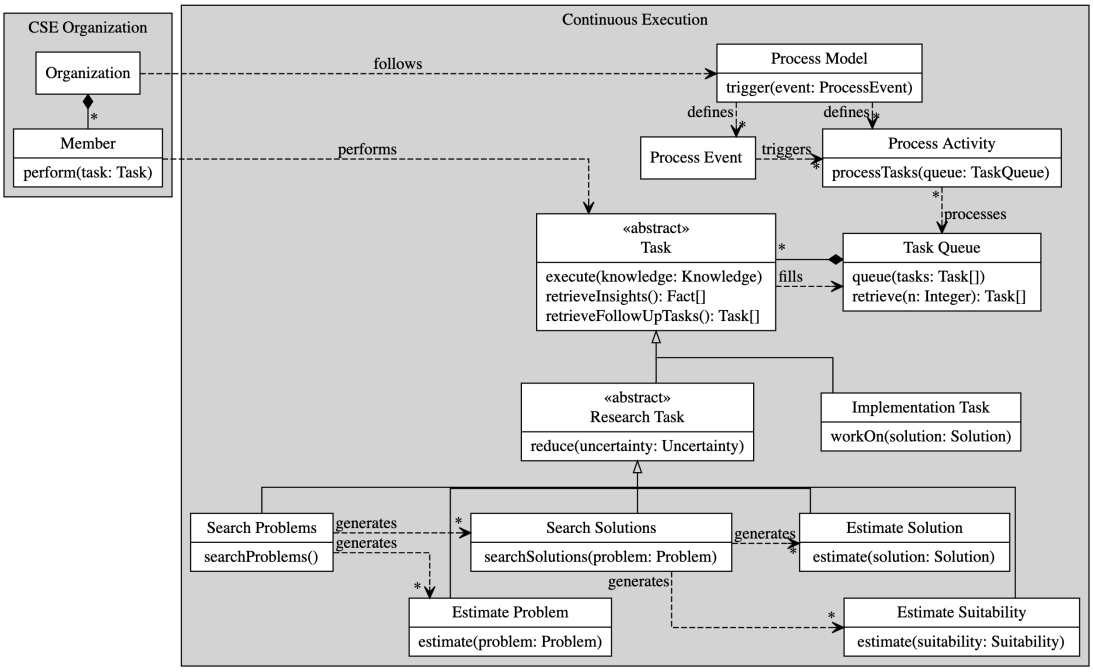


Figure 2.13: Continuous execution in CRDM based on CSEPM [74], extended with task types to fit activities across continuous delivery, experimentation, and innovation.

Continuous Delivery CSE tightly integrates implementation, verification, and delivery, creating the ability to continuously deploy new software increments to a target environment [7, 8, 9]. Continuous R&D presumes that the methodological and technological preconditions for continuous delivery, such as lean management and continuous integration, are in place. CRDM therefore simply defines a **Implementation Task** that works on implementing a solution, implying all necessary measures to integrate, verify, and deliver the implemented change. This notably does not imply that the solution will be fully implemented with a single task. Instead, a solution can require an arbitrary number of implementation tasks, as apparent in the CRDM core model.

Continuous Experimentation and Innovation Fitzgerald and Stol [1] argue that CSE requires "a feature analytics capability whereby a business manager can systematically identify a feature or set of features and quickly experiment with delivery of those features, the cost of their delivery, the usage by customers, and the actual return on investment from these features", This corresponds to the final stage of the Stairway to Heaven model for CSE maturity [99] as well as hypothesis-driven entrepreneurship approaches such as Lean Startup [15, 32]. Based on implementation and delivery, the organization can conduct continuous experimentation and establish feedback cycles [25, 9, 29]. Continuous innovation uses the stream of insights to stay responsive to evolving market conditions across the entire CSE lifecycle [99, 1]. CRDM incorporates these mechanisms of exploration and exploitation through an ontology of **Research Task**s: **Search Problems** tasks mark the exploration of the problem domain, be it by actively researching or passively receiving input from stakeholders. Problem search is followed up by both **Estimate Problem** tasks to analyze the discovered opportunities and **Search Solutions** tasks to look for options to exploit them. Similarly, this generates **Estimate Solution** tasks as well as **Estimate Suitability** tasks to analyze the identified solutions regarding their cost aspects and the cost/benefit ratio with regards to problems. These mechanisms allow CRDM to also incorporate continuous improvement (small incremental gains over time) which is enabled by in-depth understanding of problem and solution domain [7, 9].

Scientific Research In this context, continuous experimentation applies the scientific method, which itself can be described as an ongoing, iterative process[6]. By treating facts and uncertainty as questions that require constant investigation[7], Continuous R&D allows to systematically validate ideas, enable continuous learning, and reduce uncertainty regarding opportunities or return on investment [15, 30]. CRDM integrates the scientific method to allow building theories about opportunities and cost/benefit ratios, investigating phenomena, and testing assumptions to build evidence as opposed

[6] T. Garland. *The Scientific Method as an Ongoing Process*. 2015. URL: `http://idea.ucr.edu/documents/flash/scientific_method/story.htm`.

[7] A. Cho. *IBM Cloud Garage: Hypothesis-driven development*. 2013. URL: `https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development`.

to specifying fixed requirements[8]. As depicted in fig. 2.14, each research task conducts a **Research Activity** that generates the **Insight**s required to address the uncertainty the research task is concerned with. Conducting the research activity is based on frameworks for scientific research by Creswell [103], Easterbrook et al. [104], and Wieringa and Moralı [105]. Each activity follows a **Research Objective** that ask a specific type of research question (**Question Type**) which subsequently guides the research. From this, the researcher derives a **Research Strategy** that can dictates how to approach the research question, gather evidence, and evaluate it (**Strategy Type**). Accordingly, the strategy aids in selecting suitable **Research Approach**es, consisting of a **Research Method**) to conduct the research and an appropriate **Inference Method** for interpreting the results.

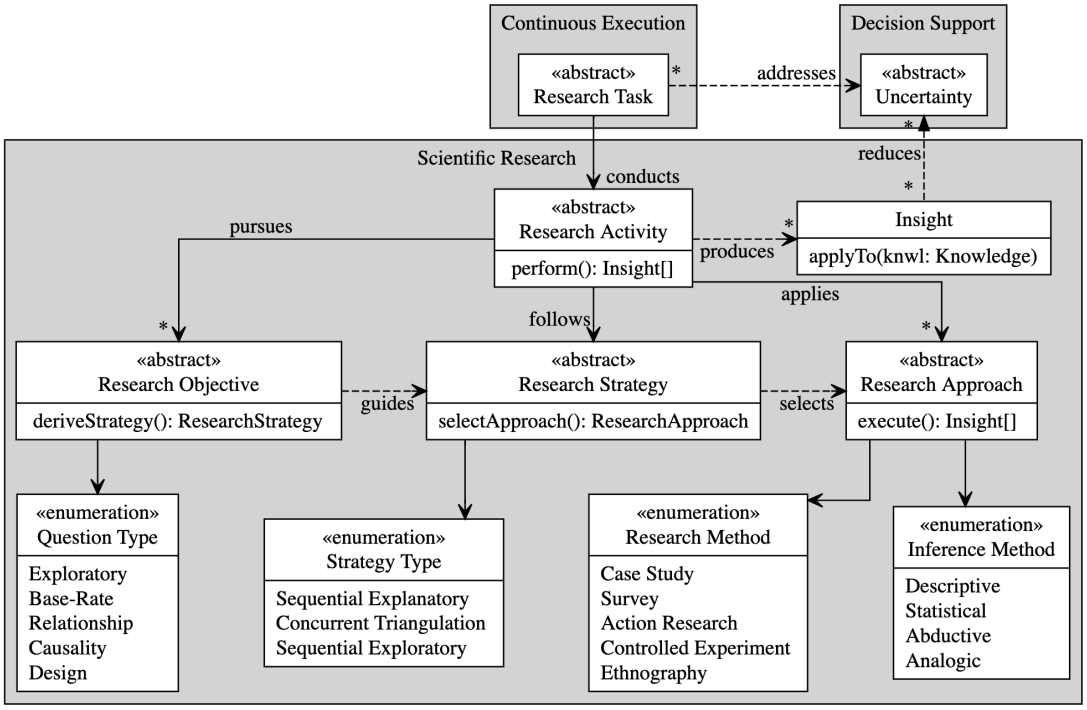


Figure 2.14: Research process generating insights to reduce uncertainty with an ontology of objectives, strategy, and approach based on [103, 104, 105].

[8] B. O'Reilly. *How to Implement Hypothesis-Driven Development.* 2014. URL: `https://www.thoughtworks.com/insights/articles/how-implement-hypothesis-driven-development`.

2.2.3 Dynamic Model

According to the OOSE model from Bruegge and Dutoit [36], the dynamic model describes the behavior of the system and assigns responsibilities to the entities identified in the object model. While doing so, new entities, attributes, and associations may be discovered and added to the analysis model. Figure 2.15 shows the fundamental sequence of activities in CRDM: Capturing of problems by both organization members and stakeholders, leading to the delivery of a solution from the organization to the customer, followed by capturing feedback from the customer to the organization.

However, this sequence of events does not imply an overall linear process. As depicted in fig. 2.16, CRDM actually expects loops through both the capturing and the delivery part. Problem capturing is expected to occur as long as there are unmet needs of customers. Until this is achieved, the organization will iteratively investigate the problem space and capture more and more knowledge about it. Solution delivery, always meant to include capturing of feedback, is expected to occur as long as there are unsolved problems. Until this is achieved, the organization will iteratively investigate the solution space, identify suitable solutions to currently known problems and deliver them to customers. It is important to note that these loops are loosely coupled, meaning that both problem capturing and solution delivery can occur asynchronously and even in parallel. This enables the organization to learn more about its domain and actually deliver solutions at the same time, in contrast to linear approaches where the learning phase strictly precedes the delivery phase.

To further refine the dynamic model, fig. 2.17 shows the use cases from the CRDM functional model as activities embedded in the fundamental activities of problem capturing and solution delivery. Their sequence shows the anticipated flow of activities but again, this does not imply a linear process. Instead, CRDM assumes that all of these activities may also occur asynchronously, allowing them to loop independently, from and to any point in the process, as often as required. This asynchronous nature of CRDM allows the organization to freely allocate their energy and focus to where it's needed at any point in time. This continuous and seemingly unorganized approach to problem-solving has been described as "chaordic learning"[9] and has been shown to enable self-organization and self-guided learning, fostering innovation and creativity in the organization [106, 35, 22].

[9] The term "chaord" is a neologism of "chaos" and "order" and means a balance of these two states that emphasizes the benefits of each [106].

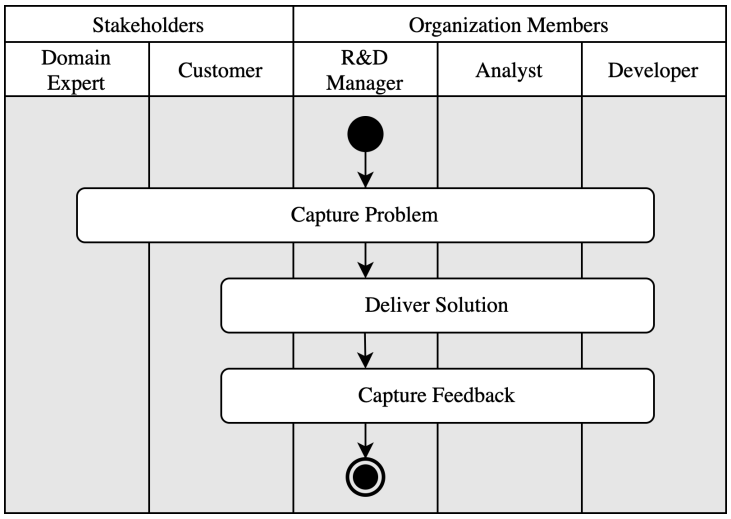


Figure 2.15: Fundamental sequence of activities in CRDM.

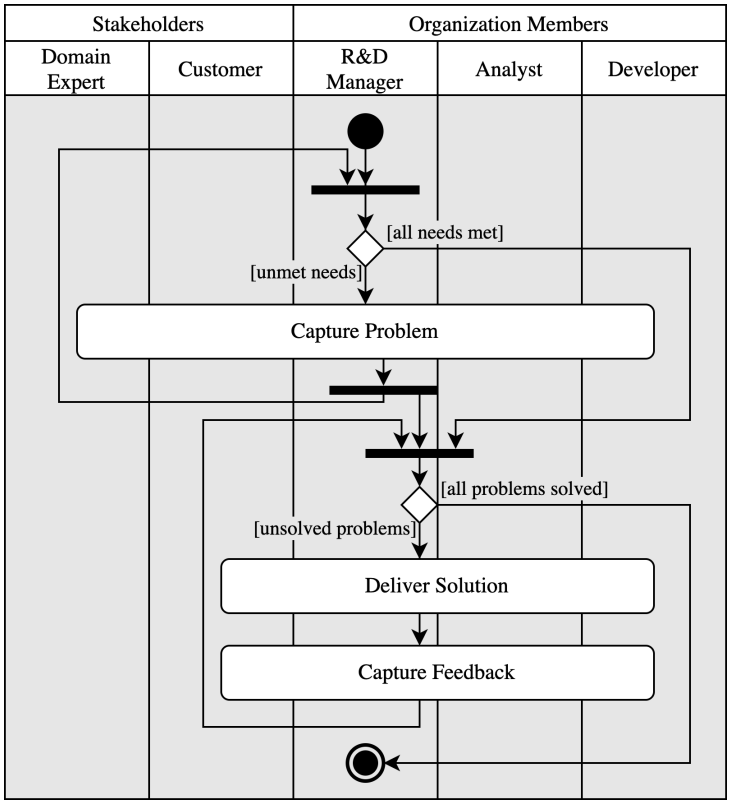


Figure 2.16: Asynchronous loops in CRDM, decoupling problem capturing from solution delivery combined with feedback capturing.

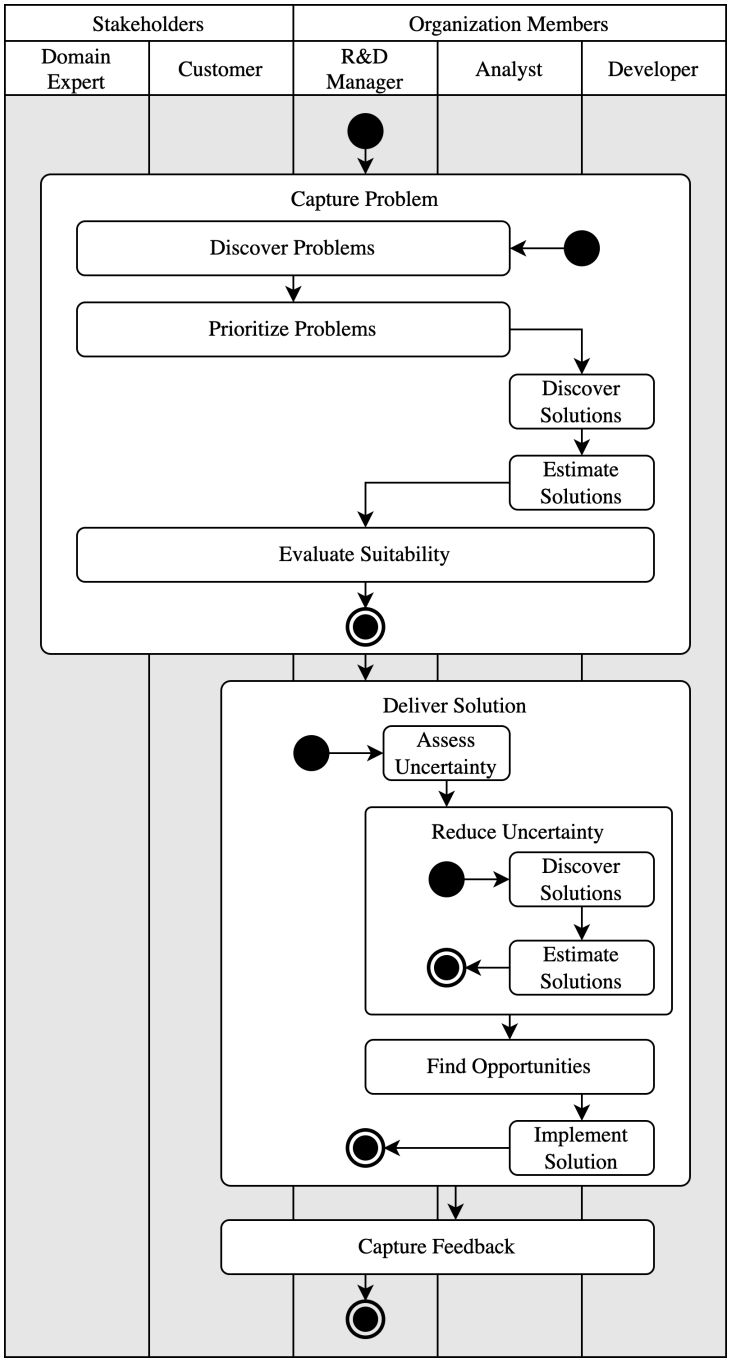


Figure 2.17: Use cases of CRDM embedded in the fundamental activities of problem capturing and solution delivery. Asynchronous loops depicted in fig. 2.16 are omitted for readability.

2.3 Validation of CRDM Applicability

The design cycle for CRDM concludes by validating the designed treatment. This is reflected by knowledge question 3: Is CRDM applicable in the context of Continuous R&D? Wieringa [63] describes different questions to ask during validation, such as which effects are caused by the interaction between artifact and context, whether the artifact satisfies its requirements, which trade-offs have to be made regarding its design, and how sensitive its performance is to context changes. OOSE also recommends reviewing and validating the results of requirements elicitation and analysis with respect to correctness, completeness, consistency, and clarity [36]. Model validation in this case means validating the initial version of the CRDM Analysis Model, specifically the dynamic model, functional model, and analysis object model. This validation therefore covers the following research questions:

1. Are the collected requirements reflected in the context of Continuous R&D?
2. Is CRDM consistent with the requirements of Continuous R&D?
3. Are all actors able to understand and apply CRDM?
4. Does CRDM fit all Continuous R&D use cases?

Validation asks the question "Are we doing the right thing?" and tries to find objective evidence that a subject meets the prerequisites for a specific application or use. Formally it is defined as "the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers." [107] In this context, it validates which complications and effects we can anticipate by observing the interactions between the artifact (CRDM) and context (CSE projects) [63]. This validation uses analogic inference to allow for efficient execution. Figure 2.18 visualizes how this approach uses simplified but adequate representations of artifact and context and then generalizes the drawn conclusions. Section 2.3.1 describes the technical action research (TAR) case study utilizing hypothesis-driven development in university projects. Section 2.3.2 presents the findings on applicability of the CRDM model as well as practical implications for projects applying it.

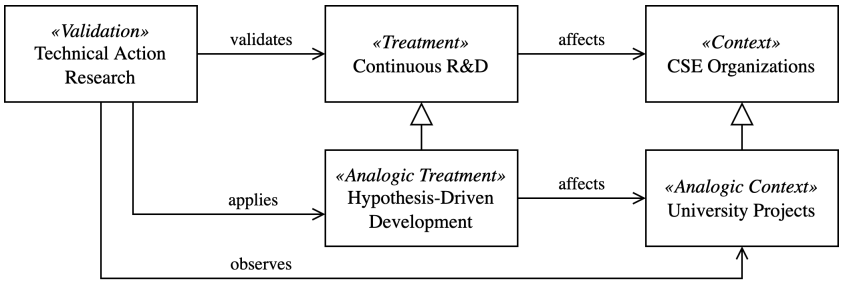


Figure 2.18: Validation of CRDM using hypothesis-driven development in university projects.

2.3.1 Case Study Methodology

Technical action research (TAR) investigates the research questions using a real-world implementation of the artifact in a real-world context. TAR intends to bridge the gap between idealized conditions of design science (the problem is framed, a suitable example of a problem class, stable, and has unambiguous goals) and the practical conditions in research projects. Instead of the usual approach of action research, starting with a concrete but idealized problem instance and reflecting on the observations when trying to solve it, TAR uses the artifact as the starting point and applies it under practical conditions to better understand both the problem class and the treatment's characteristics [105]. In TAR, researchers play multiple roles that must be differentiated: they develop the artifact, they investigate the artifact in a real-world context, and they help a client by applying the artifact. In summary, TAR uses analogical models to represent a complex phenomenon in the real world by another, more understandable or easier to analyze system. Inference design in TAR is therefore analogic, applying "generalization by similarity", and can rely on feature-based similarity and architectural similarity to make this generalization reliable [63].

CRDM validation uses six projects in a university capstone course on applied software engineering[10] as the analogical context and hypothesis-driven development as the analogical process. Validation is carried out as a quasi-experiment (without a control group) but the capstone course environment allows for validation under well-defined parameters. The following paragraphs describe suitable analogical representations for the context (CSE organization) and treatment (CRDM) as well as their implementation in the research setting of CRDM validation.

Analogical Projects The problem context is represented by six university projects (four teams with 6-8 members each and two teams with 5-6 members each). Projects are selected to be of similar size, scope, duration, structure, and process (feature-based similarity) as well as having similar problem statements, capabilities, and limitations (architectural similarity). Figure 2.19 depicts the distribution of project size, fig. 2.20 depicts the roles and responsibilities within each project. The capstone course introduces realistic project conditions by using actual problem statements from real industry customers, concrete deadlines for deliverables, and state of the art processes and tools. The project life cycle is based on a university semester: Over 12 weeks it covers introduction of a problem statement followed by requirements analysis, designing and prototyping a solution, collecting feedback from customers and field tests, and iteratively improving the solution.

Analogical Process The treatment is represented by hypothesis-driven development[11] which captures the core concepts of CRDM: continuously testing critical assumptions to

[10] iPraktikum, Chair for Applied Software Engineering, Institute for Informatics, Technical University of Munich (TUM), `https://ase.in.tum.de/projects`

[11] A. Cho. *IBM Cloud Garage: Hypothesis-driven development.* 2013. URL: `https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development`.

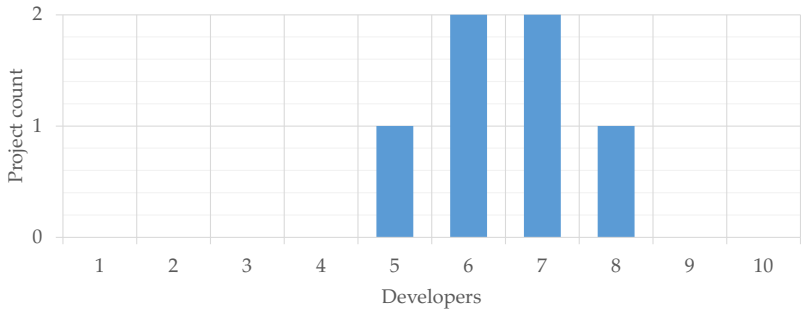


Figure 2.19: Distribution of size of selected capstone course projects (feature-based similarity).

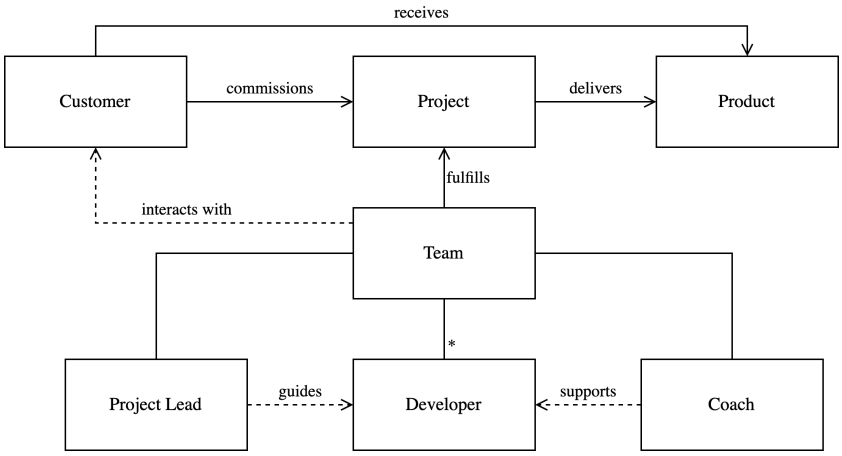


Figure 2.20: Roles and relationships in each capstone course project (architectural similarity).

generate insights for decisions based on these assumptions [32, 108]. Using an analogical process reduces the likelihood that observed effects stem from peculiarities of the process framework that are not (yet) adequately understood. The CRDM dynamic model is instantiated through hypothesis-driven design by focusing on three use cases: Evaluating feature selection, comparison of solution variants, and performing asynchronous experiments. Figure 2.21 visualizes the process for these research activities modeled after CRDM and highlights the parts that are prescribed to the projects while execution is tailored to each individual research activity.

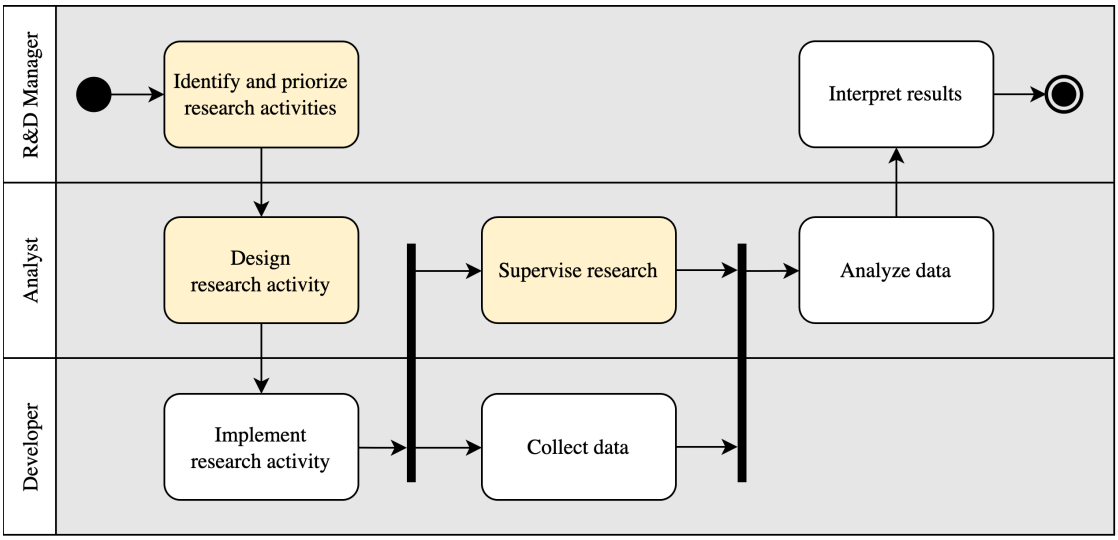


Figure 2.21: Process for research activities modeled after CRDM; validation artifact specifies highlighted parts, execution is tailored to individual context.

Each of the projects use Atlassian Jira[12] for project management and Atlassian Confluence[13] for knowledge management. For CRDM validation, projects are provided with special Jira issue types that can be linked with Confluence pages to capture additional information. This customization is achieved with a custom `WorkflowScheme`, `FieldConfigurationScheme`, and `IssueTypeScheme`. Figures B.1 and B.2 show how Jira's object model is extended with custom issue types, fields, screen schemes, and screens. Figures B.3 and B.4 show the resulting views of the custom entities. Members of the selected teams were trained to use these custom features in an introductory course before being expected to use them in their own projects.

2.3.2 Insights on Applicability

To establish a baseline for the applicability of CRDM in the context of CSE projects, projects were interviewed and described their decision processes without any knowledge of CRDM. Figure 2.22 shows the distribution of answers across the following options:

[12] `https://www.atlassian.com/software/jira`
[13] `https://www.atlassian.com/software/confluence`

- Management decision: Let the project lead decide unilaterally.
- Build to order: Rely on proposals/decisions from customer.
- Trial and error: Try one way and, if it fails try another.
- Informal experimentation: Perform something akin to experiments, but not necessarily formal.
- Preliminary research: Perform research on all variants, then select.
- Other: Use other methods to collect information for decision support.

Decisions were largely left to the R&D manager or the customer or a "trial and error" approach was applied: A variant was chosen without any defined criteria and if it caused no problems the project proceeded, otherwise a new variant was chosen. Some team members reported that they would apply informal experimentation such as benchmarks. However, they also reported that they would often choose the variant that promised to be easy to implement and offered the most options. Team members also tried to learn from the lessons of previous decisions but could not describe a structured way of doing so.

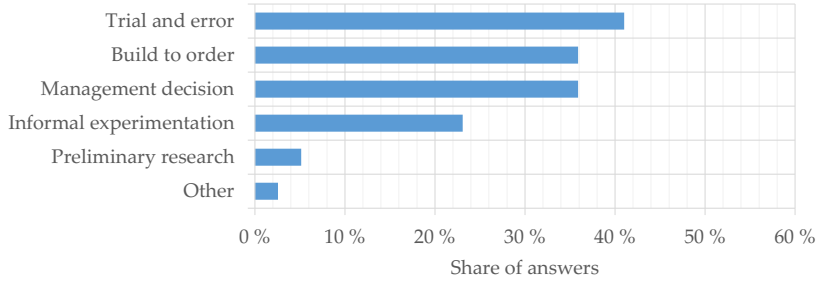


Figure 2.22: Distribution of previously used decision-making processes in capstone course projects.

Projects also described which problems they encounter when making decisions using these processes. Figure 2.23 shows the distribution of answers across the following options:

- Lack of experience: Never faced decision-making problems like that.
- Lack of information: No background knowledge about all variations involved in the decision.
- Lack of structure: No clear structure for how to approach or finalize the decision.
- Other: Other problems with decision-making process.
- None: No problems with decision-making process.

The primary problems reported by projects were a lack of information such as background knowledge about available solution variants or the specific requirements for a feature to be developed. Additionally, projects reported a lack of structure both when preparing decisions and when making decisions. This resulted in all participating teams having difficulties when faced with decisions that could impact the project's success. In

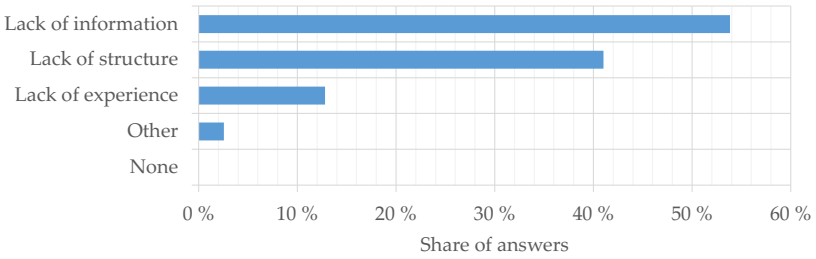


Figure 2.23: Distribution of previous problems with decision-making in capstone course projects.

these cases, a lack of experience with assessing the situation and weighing up options was reported as a further complication.

The concepts of CRDM were then introduced using a customized Jira setup towards the middle of the project lifecycle. 63 % of team members were able to participate in research activities, 44 % were able to carry out more than two research activities during the project. Even those team members who did not actively participate in research activities were able to observe the effects in their projects. Research was primarily applied in two types of situations: Firstly, design decisions were supported using methods like mockups and user testing. Secondly, technical decisions were supported through prior research and experiments such as benchmarks or live tests.

After finishing the project, members were interviewed about their experience with applying the new methods and their perceived benefit. This is subjective feedback from project members with a varying degree of expertise in software engineering methodology but it reflects the research question: CRDM is deemed applicable if these project members are able to understand it, apply it, and observe a positive effect in their decision-making. Figure 2.24 shows the distribution of answers across the following options:

- Structured approach: Research tasks give a clear structure for supporting decisions.
- Decision support: Insights from research and experiments simplify the decision.
- Managing variability: Explicit research activities encourages the team to consider all available variants.
- Demonstrating variability: Results from research and experiments allow showing different variants to the customer.
- Goal setting: Structure of the experiment and evaluation tasks encourages the team to consciously set goals.
- Other: Other benefits for decision-making process.
- None: No benefits for decision-making process.

On the one hand, participants voiced concern that research would use additional time and that following a clear structure for preparing, making, and following through with each decision would be overhead. However, this is in contrast to both the preliminary reports of a lack of structure and the overall positive sentiment about the structured approach. This indicates that there is a limit to the amount of detail that a team is willing to follow in decision support. On the other hand, participants appreciated an explicit way for setting the goals for a decision, communicating decisions, and tracking project

decisions over time (hinting at rationale management). Explicitly captured knowledge also allowed managing variability in the knowledge base and demonstrating it to the customer for further alignment (hinting at variability management).

Overall, 78 % of team members are convinced that the structured research tasks offer a benefit to the project while 22 % remain skeptical. One interesting remark in this context was that the custom Jira issue types were overhead and could just be replaced with the default tasks. However, this was only after the custom issue types were used to introduce the new concepts and learn them through practical application. This indicates that a different level of detail is required for training users on a new model compared to its ongoing application with growing experience. Finally, CRDM provided participants with a clear vocabulary of terms that enabled conversations both about the concepts and their application, regardless of the previous level of experience with structured decision-making.

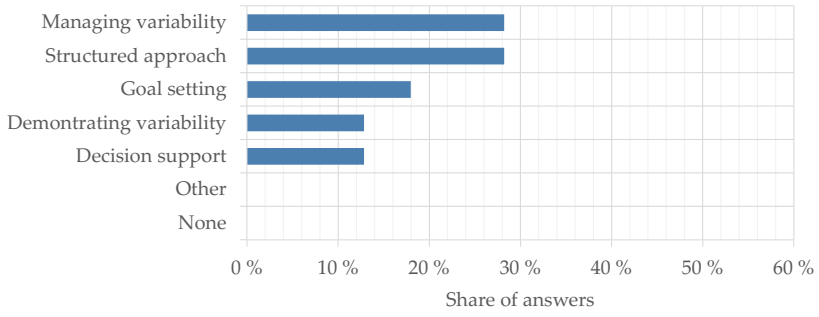


Figure 2.24: Benefits of applying the CRDM model in capstone course projects.

3 CORTEX: A Process Framework for Continuous R&D

The second design cycle of this dissertation pursues artifact design goal 1 and prediction goal 1: Designing a process framework for scientific problem solving in CSE based on CRDM and predicting its impact in CSE projects. The result is the CORTEX process framework (COntinuous Research and developmenT through EXperimentation) that guides research and decision-making in the context described by CRDM. CORTEX builds on the functional and object model of CRDM and expands the dynamic model with detailed workflows that can be tailored to each project.

Figure 3.1 visualizes how the combination of design science and OOSE methodologies addresses the design goal and prediction goal by investigating their associated knowledge questions and solving the design problem. Section 3.1 investigates the context of the design problem by surveying and mapping related process models for Continuous Innovation and Experimentation. Section 3.2 conducts treatment design by building on top of CRDM and utilizing the insights into related work, yielding the CORTEX process model and addressing the artifact design goal. Section 3.3 performs treatment validation using a simulation experiment and focuses on requirements satisfaction, design trade-offs and context sensitivity. Section 3.4 performs treatment validation using technical action research and focuses on the effects and performance of CORTEX in the project context, addressing the prediction goal. In OOSE terms, this is an iteration of analysis, leading to a refined Analysis Model, followed by corresponding model validation.

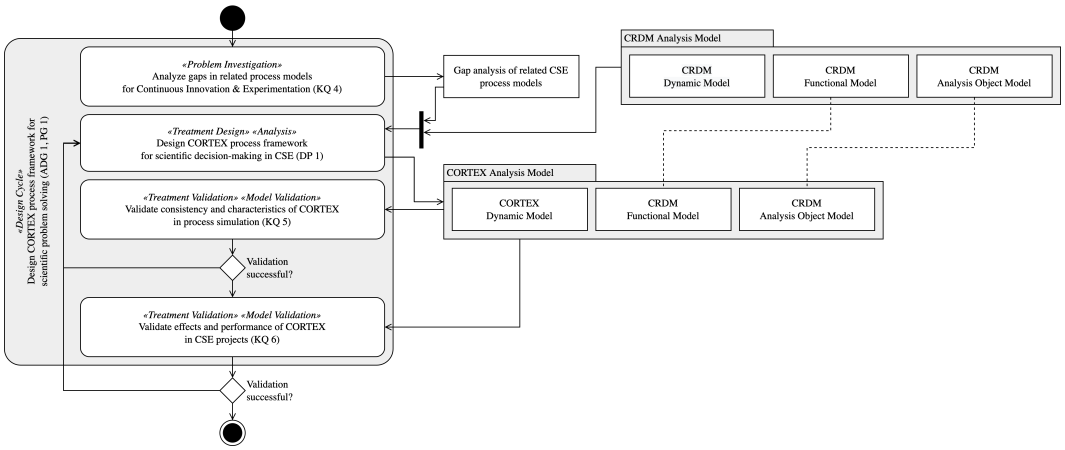


Figure 3.1: Overview of activities and entities in the second design cycle of the research project, yielding CORTEX process model and a prediction about its effects.

3.1 Gap Analysis of Related Process Models

CORTEX is built on the analysis model of CRDM and particularly extends its dynamic model. However, in order to sensibly design this next iteration of the treatment, its context needs to be further investigated and understood [63]. Furthermore, OOSE explicitly highlights systematic reuse not only of code, but also of designs and processes to reduce effort and risk while increasing standardization and reliability [36]. Based on these principles, this section conducts problem investigation within the design cycle for CORTEX by focusing on related work in process models. It investigates knowledge question 4: Which existing process models fulfill the requirements from the Continuous R&D ecosystem and could therefore serve as references for CORTEX? This analysis poses the following research questions:

1. Which approaches already exist on the matter of problem solving and decision-making in the context of CSE?
2. Where are gaps in how these process models cover the aspects of Continuous R&D as described by CRDM?

Figure 3.2 depicts how the Comprehensive Literature Review (CLR) framework [64] is used to collect and analyze related work on the subjects of continuous innovation and continuous experimentation. Section 3.1.1 describes how related process models are collected through an integrative literature review and qualitative analysis to synthesize results [65]. Section 3.1.2 presents the collected process models and categorizes them as either Continuous Innovation or Continuous Experimentation. Section 3.1.3 performs a gap analysis regarding important dimensions of Continuous R&D and identifies a disconnect between existing models for CORTEX to address.

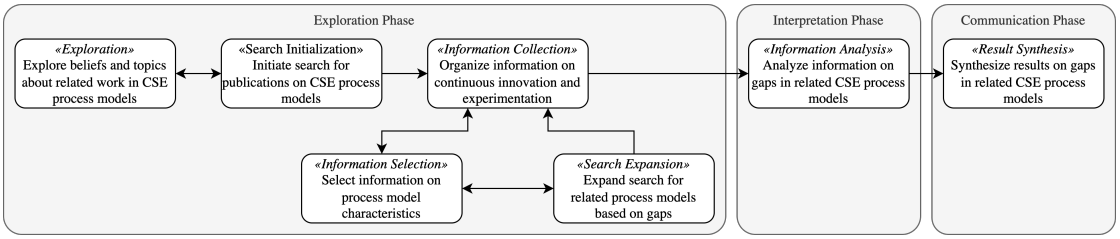


Figure 3.2: Process of literature review on related process models for problem solving and decision-making in CSE using the Comprehensive Literature Review (CLR) framework [64].

3.1.1 Mapping Study Methodology

To answer the knowledge question, the mapping study employs a two-step process: A **literature review** identifies existing models for decision-making in the CSE ecosystem. Specifically, the activities of *continuous innovation* and *continuous experimentation* are chosen as starting points. These two topics contain the relevant decision processes that produce input for other activities such as continuous planning and continuous delivery or use information from other activities such as continuous monitoring. Models are accordingly collected them into groups according to their focus to ensure that the subsequent mapping step accounts for differences between models. Subsequently, a **mapping step** assigns the models to relevant dimensions of decision-making. Within each dimension, each model is rated according to how comprehensive their coverage of that dimension is on a numerical scale. This makes coverage a numerical attribute instead of a binary one and allows to better assess overall coverage of all dimensions of decision-making. Finally, a **gap analysis** assesses the combined coverage of all models across all dimensions to identify the need for further research in this area. The literature review also generates the categories for mapping and gap analysis using qualitative coding. Table 3.1 lists these categories here preemptively for a better overview.

Category	Description
Continuity	Execute activities asynchronously (ideally in parallel) and iteratively.
Opportunism	Learn from unexpected insights and seize opportunities for value creation.
Experimentation	Test hypotheses, measure effect of changes, compare alternatives.
Prediction	Validate critical assumptions before investing significant time and resources.
Innovation	Discover market conditions, customer needs, potential features, or technologies.
Refinement	Improve and optimize existing solutions on performance, quality, utilization, etc.
Decision-making	Support decisions regarding priorities, feature set, technology etc.
Research methodology	Use research method and data source appropriate for the situation.

Table 3.1: Categories for mapping and gap analysis of CSE process models.

3.1.2 Process Models for Innovation and Experimentation

Hypothesis-driven development (HDD) is an identifiable trend towards scientific problem solving in CSE and as such an underlying principle of all process models presented here. The hypothesis-driven approach to decision-making in commercial projects originates from the "lean startup" movement popularized by Ries [15], who further refined the methodology in collaboration with Eisenmann et al. [32]. They provide guidelines for formulating and evaluating hypotheses as well as acting on the results of the hypothesis tests. A hypothesis in this context is any assumption that is critical to project success and is falsifiable by means of an experimental method. Identifying and testing these critical assumptions is used as a guiding principle for important decisions of the project. Figure 3.3 visualizes the essential workflow of an hypothesis-driven approach.

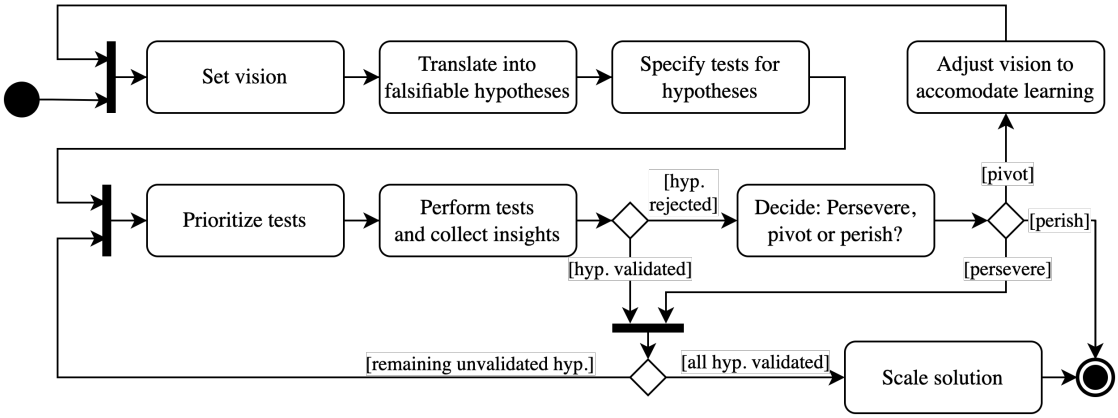


Figure 3.3: Process for hypothesis-driven development (adjusted from [32])

Other sources also provide guidelines for dealing with complexity and uncertainty by using HDD as an enabler for data-driven decision-making.[1] In this case, HDD formulates hypotheses about whether a specific action will improve a critical metric. The key for successful application of this approach is choosing the right metrics to optimize for and avoiding to target the wrong ("vanity") metrics Ries [15]. This approach has found its way into empirical software engineering and multiple models have been proposed to apply the scientific method before proceeding to solution engineering and product development. They define hypotheses for assumptions about business strategy, feature value, or feature refinement, depending on whether their focus lies on initial exploration or continuous development. For data collection they employ techniques like customer feedback, prototypes, or analytics data from live experiments [33, 10, 35, 28, 29]. In this context, it is important to differentiate between high-level decision processes and specific

[1] A. Cho. *IBM Cloud Garage: Hypothesis-driven development*. 2013. URL: `https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development`, J. L. Taylor. *Hypothesis-Driven Development*. 2011. URL: `http://www.drdobbs.com/architecture-and-design/hypothesis-driven-development/229000656`, B. O'Reilly. *How to Implement Hypothesis-Driven Development*. 2014. URL: `https://www.thoughtworks.com/insights/articles/how-implement-hypothesis-driven-development`.

mechanics of experimentation. The focus of this literature review is decision support for decision problems such as triage, prioritization, selection, or scheduling [80, 37].

Process Models for Continuous Innovation The literature review collected process models that fit the definition of continuous innovation and are focused on innovation and value prediction. These models attempt to tackle the problem of the "open loop" between company and customers. As assumptions about requirements and priorities are often wrong and customers themselves often cannot express their needs precisely, a hypothesis- and data-driven approach to finding the right problems to work on and solutions to apply reduces uncertainty, supports decision making and helps to develop valuable products [110, 30, 31]. The following list presents the selected models along with their description as given by their respective authors. Table 3.2 identifies the primary areas of focus for these process models.

- **Hypothesis-Driven Entrepreneurship (HDE)** described by Eisenmann et al. [32]: "A hypothesis-driven approach helps reduce the biggest risk facing entrepreneurs: offering a product that no one wants." "A lean startup may eventually invest enormous amounts of capital in customer acquisition or operational infrastructure—but only after its business model has been validated through fast and frugal tests."
- **Innovation Experiment System (IES)** described by Bosch [25]: "The loop between deploying new functionality, measuring usage and other performance metrics and subsequently using the collected data to drive development is the main process", where "the basic principle is that we want to invest as little possible until deploying something to customers."
- **Early Stage Startup Software Development Model (ESSSDM)** describes by Björk et al. [111]: "The model extends already existing lean principles, but offers novel support for practitioners for investigating multiple product ideas in parallel, for determining when to move forward with a product idea, and for deciding when to abandon a product idea."
- **Early Value Argumentation and Prediction (EVAP)** described by Fabijan et al. [34]: "a technique that practitioners can use in order to estimate what impact a feature will have when fully developed" so that it "works as a support for helping companies move away from early specification of requirements and towards dynamic feature prioritization".

Process Models for Continuous Experimentation The literature review collected process models that fit the definition of continuous experimentation and are focused on experimentation and continuous execution. These models attempt to close the "open loop" through feature experimentation and capitalizing on customer feedback to guide the development process. Models addressing this problem aim to gather feedback accurately

Process model	Focus area(s)
Hypothesis-Driven Entrepreneurship (HDE)	Decision-making, Innovation
Innovation Experiment System (IES)	Research methodology
Early Stage Startup Software Development Model (ESSSDM)	Innovation
Early Value Argumentation and Prediction (EVAP)	Prediction, Decision-making

Table 3.2: Identified areas of focus of related process models for continuous innovation.

and timely, confirm feature value, enable data-driven decisions and mitigate the risk of delivering wrong products [99, 28]. These models also emphasize the continuous execution of activities, achieved through the definition of small bundles of activities that can be embedded in larger processes. However, theses processes are still defined as linear sequences of events and not on the basis of asynchronous events. The following list presents the selected models along with their description as given by their respective authors. Table 3.3 identifies the primary areas of focus for these process models.

- **Hypothesis Experiment Data-Driven Development (HYPEX)** described by Bosch [10]: "The HYPEX model was created to close the 'open loop' between customers and product management" and "describes how companies can run feature experiments to close the 'open loop' in order to confirm that a feature that was selected for development has the expected value."
- **Qualitative/Quantitative Customer-driven Development (QCD)** described by Olsson and Bosch [35]: "The model identifies a number of customer feedback techniques (CFT's) that can be used to validate feature value with customers" whereby "hypotheses are derived from business strategies, innovation initiatives, qualitative and quantitative customer feedback and results from on-going customer validation cycles."
- **Data-Driven and Value-Oriented Continuous Experiment (DVOCE)** described by Ekström and Þorvaldsson [28]: "DVOCE is a detailed and extended version of the previously published high-level HYPEX model and covers the pre-development and development phases. DVOCE provides a detailed procedure in how to model the feature to enable the prediction."
- **Rapid Iterative value creation Gained through High-frequency Testing (RIGHT)** described by Fagerholm et al. [29]: "Building blocks for a continuous experimentation system and infrastructure" that focus on "developing the *right* software, whereas the typical focus of software engineering in the past has been on developing the software right (e.g. in terms of technical quality)."

Process model	**Focus area(s)**
Hypothesis Experiment Data-Driven Development (HYPEX)	Continuity, Experimentation
Qualitative/Quantitative Customer-driven Development (QCD)	Continuity, Prediction
Data-Driven and Value-Oriented Continuous Experiment (DVOCE)	Prediction, Experimentation
Rapid Iterative value creation Gained through High-frequency Testing (RIGHT)	Continuity, Experimentation

Table 3.3: Identified areas of focus of related process models for continuous experimentation.

3.1.3 Gaps in Existing Process Models

Fitzgerald and Stol [9] and Fitzgerald and Stol [1] describe a gap between the processes of business and development and coin the challenge of integrating them "BizDev", analogous to "DevOps" for the integration between development and operations. This gap is reflected by the gap between the analyzed process models for continuous innovation and experimentation. Additionally, the authors of these models describe remaining challenges that need to be addressed to better use them or further integrate them. To investigate this gap, the selected process models are rated regarding their coverage of the categories determined in the literature review. Rating uses a numerical scale from 0 ("category scarcely covered") over 1 ("category rudimentarily covered") to 2 ("category extensively covered").

Figure 3.4 shows the summarized ratings of all models per category and therefore illustrates how well each category is covered be the related process models in CSE. In the following graphs, the categories are already ordered according to their relation with innovation vs. experimentation topics so that the split becomes more apparent. Peaks become apparent at decision-making and innovation as well as continuity and prediction. Taking a closer look at the correlations of this coverage, fig. 3.5 plots the coverage of each process model with a transparency of 50 % on a radar chart of the categories. The areas where multiple models overlap are therefore more opaque than areas covered by fewer models. Now, two poles become apparent: Prediction, continuity, and experimentation as well as innovation and decision-making. This aligns well with both the grouping of models into innovation-focused and experimentation-focused ones. It also visualizes the gap between decision and execution processes, reflecting the "BizDev" challenge of closing the gap between business (being decision-focused) and development (being execution-focused).

Figures 3.6 and 3.7 show the rating of each model in detail. Innovation-focused models focus on testing assumptions on decisions that are critical to project success, mostly which features to work on, but largely leave out how to implement these approaches. They utilize a hypothesis-driven approach to reduce waste, increase speed, support pivoting when a test fails and deliver valuable products. Their style of using minimum viable features (MVFs) or minimum viable products (MVPs) is rooted in the "build-measure-learn" cycle popularized by the lean startup movement [15, 30]. ESSSDM touches on generating, prioritizing and validating an idea backlog, but focuses on validating MVP value in early stage startups rather than ongoing product development [33, 28]. Experimentation-focused models focus on feature validation, either predictively or through feedback, and show a disconnect from innovation and decision-making. This impression is reinforced by the fact that these models presume a feature backlog or product vision to just exist or be derived from business strategy or customer understanding by some method that is left up to the project to determine [27, 34, 28, 29]. QCD proposes using customer feedback techniques like interviews or prototyping for generating and validating product ideas or requirements but doesn't describe any detailed process other than treating them as hypotheses under test and evaluating them using qualitative or quantitative methods [35]. RIGHT aims to integrate build-measure-learn blocks for learning and

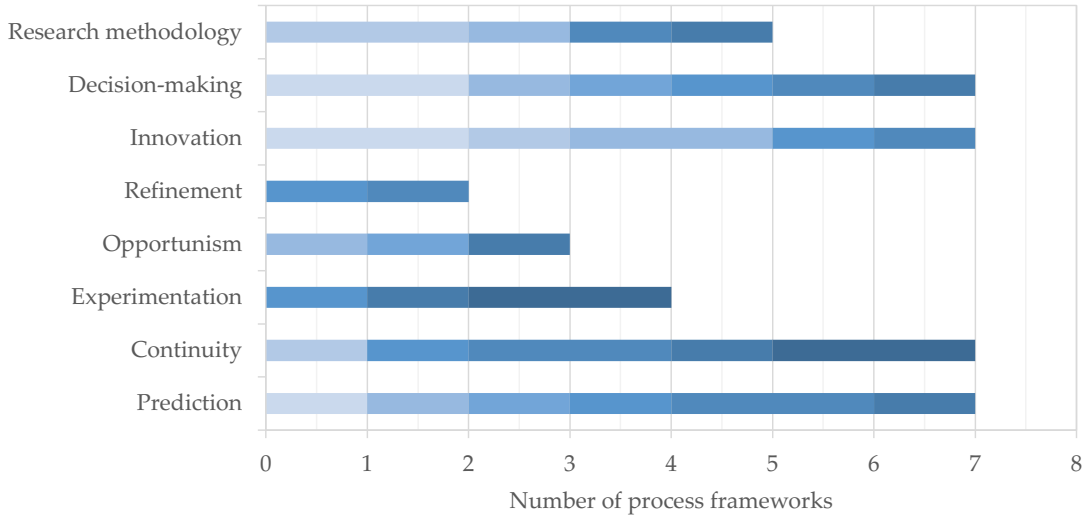


Figure 3.4: Number of process models covering each category.

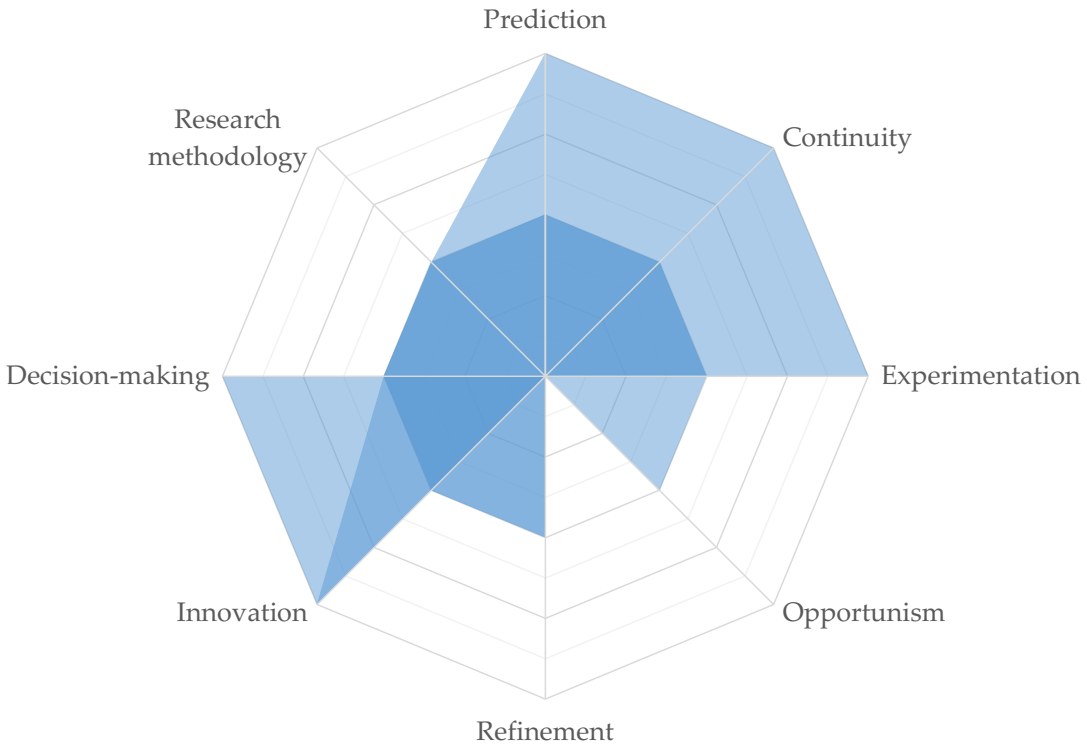


Figure 3.5: Rating of all process models on all categories overlayed to form a "coverage map" with transparency indicating the density of coverage.

decision support, with each block providing input for the next iteration [29]. However, it remains an abstract model focused on experimentation and as such provides only superficial guidelines. There are also categories that are nearly or completely untouched by the evaluated models: Guidelines for research methodology are only provided by IES, all other model only provide rudimentary hints or none at all. Only few models like QCD or HYPEX describe processes for validating assumptions or understanding context based on hypothesis tests. All models are lacking a focus on opportunism and refinement, even though both are cornerstones of lean and agile development methodologies. However, experimentation-focused models at least cover the basics of other categories as is apparent from the circle-like shape, while innovation-focused models only touch on one additional aspect as is apparent from the singular lines sticking out in those directions.

Once an environment for continuous innovation and experimentation is available, it facilitates continuous learning through experimentation and helps with making critical decisions, understand user behavior, and deliver impactful functionality while saving time and resources [15, 25, 30]. However, the reviewed literature mentions further challenges for continuous innovation and experimentation before they can yield valuable results [30, 31, 29]:

- Technical challenges:
  - Implementing infrastructure for experimentation within mature product,
  - identifying target metrics that are both valuable and measurable,
  - storing and transferring massive data from customer environments,
  - finding suitable methods with constraints like low usage volume.
- Organizational challenges:
  - Identifying and prioritizing critical assumptions,
  - finding viable experimental approaches for business environment,
  - teaching experimentation concepts before they can be applied,
  - prioritizing customer requirements opposed to innovative culture.
- Customer-related challenges:
  - Informing customers of major changes and perform acceptance tests,
  - receiving and understanding feedback from end users that are customers of customers (B2B2B or B2B2C scenarios),
  - finding a pro-active lead customer for testing ahead of widespread releases,
  - constructing legal agreements for collecting and processing data on user behavior.

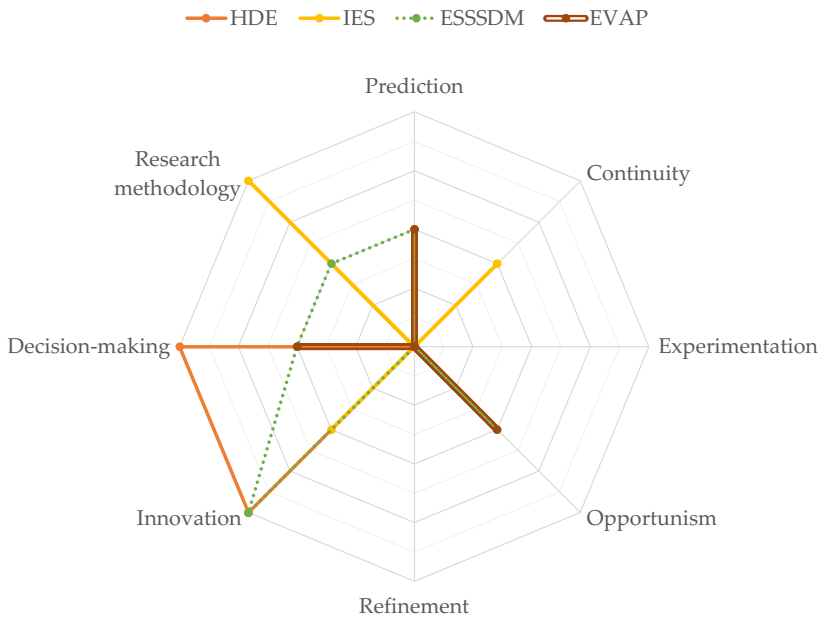


Figure 3.6: Rating of process models focused on innovation and value prediction, visualization adapted from [61].

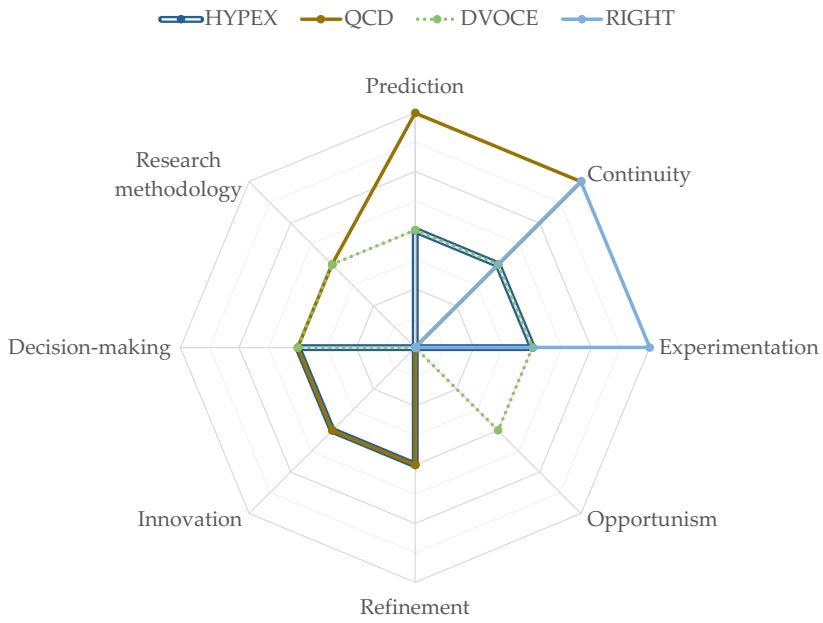


Figure 3.7: Rating of process models focused on experimentation and continuous execution, visualization adapted from [61].

3.2 Design of CORTEX Process Framework

After investigation of the problem context, the next step in the CORTEX design cycle is treatment design [63]. Using both CRDM and related process models as a foundation, it solves design problem 1: Designing a decision-making process framework that integrates scientific problem solving with CSE so that decisions can be made more effectively. This involves the following research questions:

1. How can CRDM be extended to address the gaps identified in related process models for Continuous Innovation and Experimentation?
2. How can an overarching process combine existing techniques and best practices for problem solving, research, and decision-making?
3. How can the process be simultaneously thorough but also flexible and continuous to fit with CSE principles?
4. How can the resulting process be adapted to the characteristics of a specific CSE project?

In OOSE terms, this represents another iteration of analysis, yielding the CORTEX Analysis Model which extends CRDM Analysis Model with a detailed dynamic model while leaving the functional and object model unchanged. Figure 3.8 depicts how the CORTEX process framework is designed from guiding principles over the overarching process and underlying workflows to the variation points for process tailoring. Section 3.2.1 describes the methodology for process design based on CSE principles, chaordic learning, and an existing CSE process meta-model. Section 3.2.2 presents the architecture of the Continuous R&D process that connects all activities within CORTEX as well as to surrounding CSE processes in an event-driven fashion. Section 3.2.3 describes the core problem solving process that drives the decision-making in CORTEX. Sections 3.2.4 and 3.2.5 describe the workflows utilized by the problem solving process for decision support and continuous execution. Section 3.2.6 describes options for process tailoring by adapting the individual workflows to the needs of a specific project.

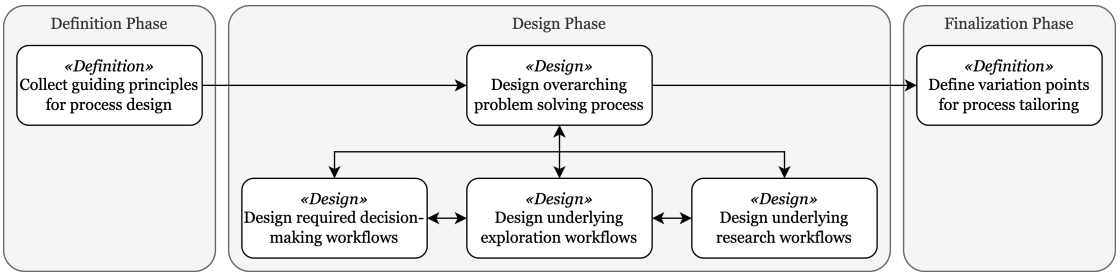


Figure 3.8: Process of designing the CORTEX process framework consisting of an overarching process and multiple underlying workflows.

3.2.1 Process Design Methodology

CORTEX is a process model that aims to enable continuous learning, empirical problem solving, opportunity seizing and decision support through continuous research activities. Before treatment design, a set of guiding principles is established to ensure that the resulting process framework conforms to the requirements of the Continuous R&D ecosystem. First, the goal of is to support different types of decision problems such as classification, prioritization or selection [41]. These decisions are to be supported using insights produced from research efforts, thereby reducing uncertainty and involved risks. In doing so, it needs to differentiate between experimentation for exploration and optimization for refinement. Second, CORTEX needs to provide both a process that continuously pursues the general goal of problem solving and workflows that can be executed to accomplish specific tasks. To become a process framework, it also needs to specify suitable variation points for tailoring the process to a specific project environment. Third, CORTEX may draw inspiration from approaches to "chaordic" organizations with an emerging instead of defined structure as well as solving "wicked" problems with no clear definition or solution. These approaches need to be adapted to the context of Continuous R&D, as elaborated in the following guiding principles.

Exploration and Exploitation CSE enables shorter development cycles by achieving completeness and consistency through incremental and iterative development, respectively [112, 26]. These techniques can be used for both exploration to discover customer needs and deal with dynamic environments and for refining existing solutions based on feedback [31]. Exploration (of problem and solution space) and exploitation (of discovered value) are therefore two complementary activities that need to be combined within the process model. This corresponds to the lean principles of radical change ("Kaikaku") and incremental improvement ("Kaizen") [1, 33]. This necessitates broadening the horizon of other definitions as well. In terms of problem solving, the process framework needs to employ a "satisficing" approach for incremental optimization instead of searching for the optimal solution at once [38, 41]. Before accepting a solution, the project needs to both validate of right problem/solution fit as well as verify the solution quality.[2]

Processes and Workflows In everyday language, the terms "process" and "workflow" are used interchangeably. However, there is an important difference that needs to be defined to accurately use the terms in process design. A process describes an abstract procedure which can consist of multiple steps, ordered in a logical way and with causal relationships between them. In doing so, it *pursues an objective* such as delivering a product, service, decision, or other. In contrast to this, a workflow gives details instructions on how to carry out a certain scenario using a set of tools, techniques, and technologies. In doing so, is *produces an output* such as an artifact, information, design, code, or other. A workflow can be part of a process but not vice versa. As such, a process

[2] S. Easterbrook. *The difference between Verification and Validation.* 2010. URL: `http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/` (visited on 11/29/2010).

can exist without defining any underlying workflows but, conversely, a workflow requires an overarching process to make sense [114].

Chaordic Systems Generating innovation while facing issues like conflicting stakeholder needs and constantly evolving domains and technologies requires continuous learning and creativity, resulting in a "chaordic" approach. The term "chaord" is a neologism referring to the state between chaos and order. A chaordic organization applies a self-organizing, adaptive learning process to achieve innovation in complex environments. These organizations are charaterized by multiple independent entities whose interactions lead to self-organization and induce learning [106, 22]. This principle is rooted in Heraclitian philosophy as "panta rhei" (everything flows)[3], viewing everything through the concept of flow and postulating the famous adage that change is the only constant [115]. This resonates with the shift of CSE away from control-based mechanism to fluid transitions between flexible processes, leading to the emergence of the desired results [1, 22].

Wicked Problems Wicked problems are a class of problems that are especially difficult or even impossible to solve due to their characteristics. Initially defined in the context of social policy planning, their definition has been adapted to the general problem solving and decision-making [66, 57, 67]. Suitable approaches such as IBIS and CONSUL are based on the concepts of issues and surrounding argumentation based on goals, facts, and positions [60, 78]. The process framework should draw inspiration from these existing approaches and adapt them to the context of CSE as needed. In doing so, it should emphasize knowledge and rationale management to make these mechanisms explicit and understandable [107, 97, 62].

[3]The working title for this process framework therefore was PANDA (Problem solving And iNnovation through Decision support and Assumption testing) as a play on "panta rhei".

3.2.2 Continuous R&D Process

CORTEX is a process framework, i.e., a process that can be adapted to a project's or organization's individual needs through so-called "tailoring". For this reason, it consists of multiple workflows that can be adapted individually. Its process architecture follows CRDM and consists of multiple workflows for problem solving, decision support, and continuous execution. Together, these workflows augment existing process models for continuous innovation and experimentation by closing the gap between business and development processes. The workflows are *asynchronous*, meaning that no specific sequence of activities is assumed. Instead, all workflows are coupled through events which they emit and to which they react.

3.2.2.1 Process Architecture

CORTEX has the goal to foster the integration of business and development processes, described by Fitzgerald and Stol [9] as "BizDev". To achieve this, it needs to enable flow of information to form a feedback loop between the CSE processes and the customer as the driving element. Figure 3.9 depicts how CORTEX uses ties together the processes of the Continuous * model (initially described in section 1.1.1): The loop is closed by connecting the requirements that business collects from the customer to the solutions that development delivers. This might sound simplistic, but it is the essential feedback loop that CSE is focused on [15, 10, 9]. This flow of information occurs on two levels:

- On the top level, CORTEX establishes a **problem solving process** that continuously captures the customer's needs and treats them as problems to be solved. It therefore also interacts with the customer directly, following the principles of agile development. It continuously searches for opportunities to solve the problems and tries to reduce the risk inherent in the decision it makes. To achieve this, it issues research and implementation tasks that are executed continuously using the organization's DevOps processes.
- The foundation for the problem solving process is therefore formed by **decision support workflows** that explore the problem and solution domain, search for connections, assess benefits or costs of entities, and so on. In doing so, they always capture the uncertainty factors inherent in these facts to later assess their confidence level. These activities are enabled by **continuous execution workflows** that utilize the CSE organization's capabilities for continuous experimentation and continuous delivery to generate insights for decision support as well as to execute on decisions.

Both levels are connected through a **knowledge base** containing all information that is relevant for decision support. This represents the "blackboard pattern" from software architecture for expert systems. In this architecture, a problem solving process is distributed across multiple specialized actors, each acquiring knowledge from different sources. Knowledge is produced and processed individually, but shared through a central datastore and coordinated by a non-deterministic control strategy [116].

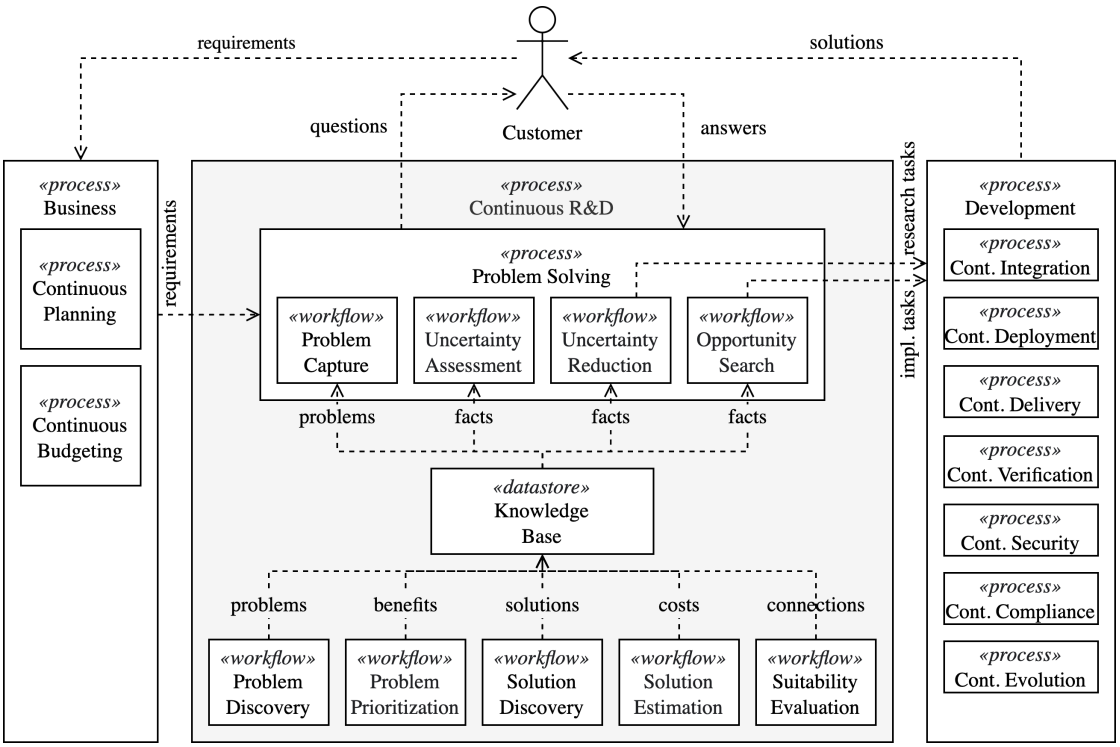


Figure 3.9: Continuous R&D processes establishing "BizDev" link between business and development (according to Continuous * model [1], operations omitted) as well as internal workflows for generating and utilizing knowledge.

3.2.2.2 Event-Driven Execution

CORTEX is an asynchronous process model based on activities that are triggered by events and in turn emit events. As the basis for this event-driven execution, CORTEX adapts the central event loop from CSEPM meta-model for continuous processes by Krusche and Bruegge [74] (initially described in section 2.2.2.3). As depicted in fig. 3.10, CORTEX executes the event loop continuously over the entire lifetime of a project. This event loop captures events emitted by the activities and triggers corresponding activities. New activities are captured as tasks in a queue that acts as a buffer and is partitioned and prioritized according to the projects needs. Tasks are executed according to the organization's process model (e.g., Scrum or Kanban) This event-driven model allows CORTEX to stay flexible and continuously react to change in a nonlinear fashion. It thereby follows the principle of chaordic learning: Given that the process framework is tailored to the organization's needs, the loose coupling of activities allows the behavior necessary for problem solving to *emerge* instead of being determined in advance.

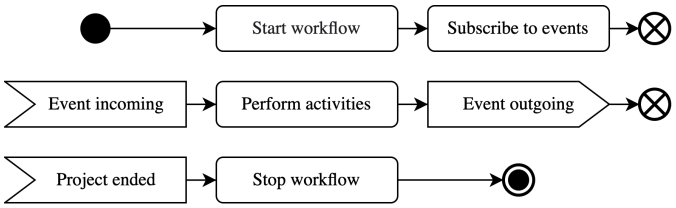


Figure 3.10: Event loop running during the project lifecycle, adapted from CSEPM [74].

CORTEX defines specific events that will trigger the corresponding activities across the problem solving process and the workflows for decision support and continuous execution. Events are grouped into packages according to CRDM:

- In problem solving, *customer events* describe the interaction with the organization's primary stakeholder, capturing their needs, delivering solutions, and capturing corresponding feedback. These events are handled directly in the problem solving process.
- In decision support, *problem domain events* and *solution domain events* cover the exploration as well as investigation of problems and solutions, respectively. These events facilitate the exchange of information between the problem solving process and the workflows for decision support. Additionally, *connection events* cover the investigation of the connections between problems and solutions.
- In continuous execution, *work queue events* describe pushing to and pulling from the queue of work items while *work item events* provide information about the status of work items. These events form the generic layer of asynchronous execution used by both research and implementation activities.

Note: Subsequent models may omit enqueuing and dequeueing of work items for brevity, but all events that trigger tasks are meant to be tied to work items.

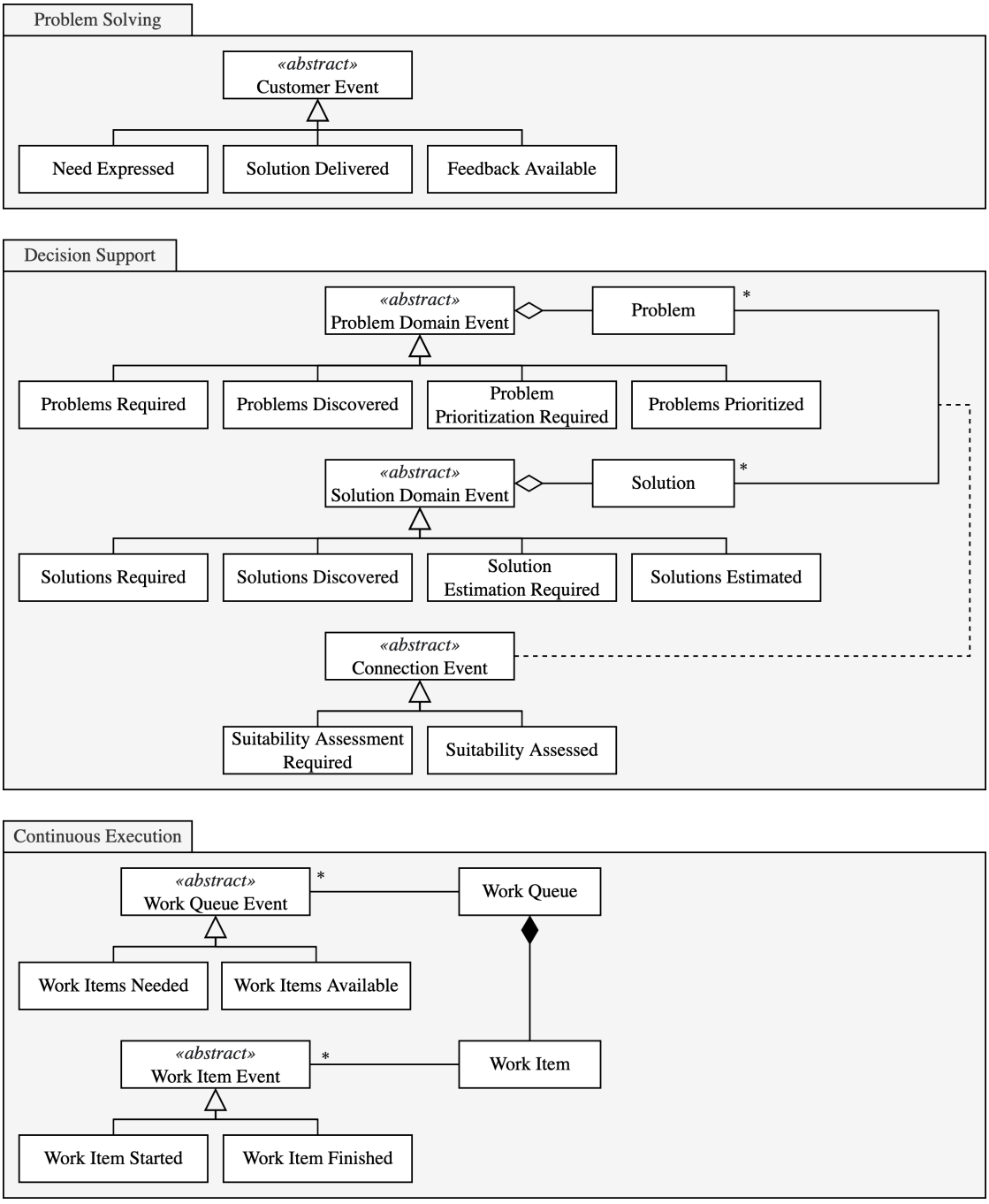


Figure 3.11: Event taxonomy of the CORTEX process model; abstract base classes associate events with entities.

3.2.3 Problem Solving Process

CORTEX differentiates interaction with stakeholders and the core problem solving process. As modeled by CRDM, stakeholder interaction is handled primarily by the R&D manager while the core problem solving process relies on research and decision support by analysts. Decisions are made using a data-driven and probabilistic approach.

3.2.3.1 Stakeholder Interaction

The R&D manager facilitates stakeholder interaction (shown in fig. 3.12) and therefore plays an important role in guiding the project in a customer-driven direction:

- Capturing needs as problems (or hints at problems) to be investigated and solved,
- eliciting feedback and updating the knowledge base as well as project parameters,
- facilitating communication around new work items (including with domain experts),
- facilitating communication around the release or update of solutions.

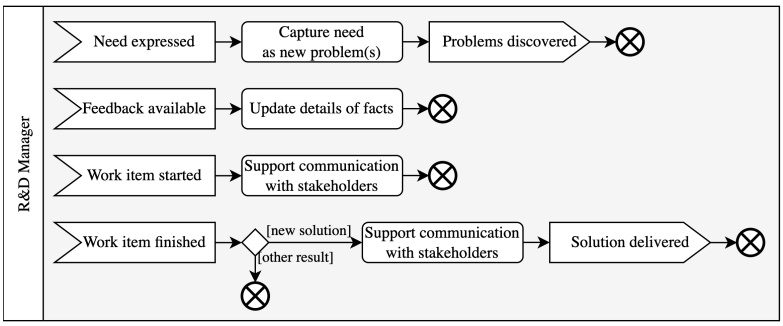


Figure 3.12: Process for coordinating the problem solving process between organization and customer.

3.2.3.2 Opportunistic Problem Solving

The CORTEX problem solving process traverses the problem space iteratively, revisiting entities to react to updated information, and recursively, moving through hierarchy and variability of problems and solutions. Problem solving is treated as a series of prioritization, selection, and classification problems [37]. These decisions are supported by invoking the decision support workflows which in turn execute research activities to answer questions or validate hypotheses. To decide which course of action to pursue, CORTEX applies a probabilistic approach: It approximates the expected value (i.e., the average value of all possible outcomes) of every possible decision. It is calculated from problem benefit, solution cost, and respective confidence P across all variations p and s:

$$\mathbb{E}[Problem, Solution] = \sum_{p,s=1}^{\infty} \frac{Benefit_p \cdot P[Benefit_p]}{Cost_s \cdot P[Cost_s]} \cdot P[Connection_{ps}]$$

This reveals opportunities to solve high-benefit problems by implementing connected low-cost solutions under sufficient confidence in those facts. Vice versa, it reveals areas where the risk will outweigh the potential reward and further research is needed before being able to make a decision. Figure 3.13 visualizes how this information is compiled and then used to generate implementation and research tasks accordingly:

1. Determine the *benefit/cost ratio* for combinations of problems and solutions.
2. Use the captured uncertainty factors to asses the confidence in the involved facts.
3. Calculate the *risk/reward ratio* of implementing/continuing each possibility.
4. Partition entities according to their confidence, yielding "hot spots" of opportunity and "cold spots" of uncertainty in the knowledge base.
5. Check pending decisions against existing research and implementation tasks (regardless of their status) to avoid redundant or conflicting decisions.
6. Create research and implementation tasks according to the new decisions.

This process is asynchronous and triggered by multiple events: It initializes the knowledge base at the start of the project, it may be executed at timed intervals (e.g., the project's sprint cycle) and it reacts to new information becoming available from research activities. Both research and implementation is then executed asynchronously and results are handled as they become available.

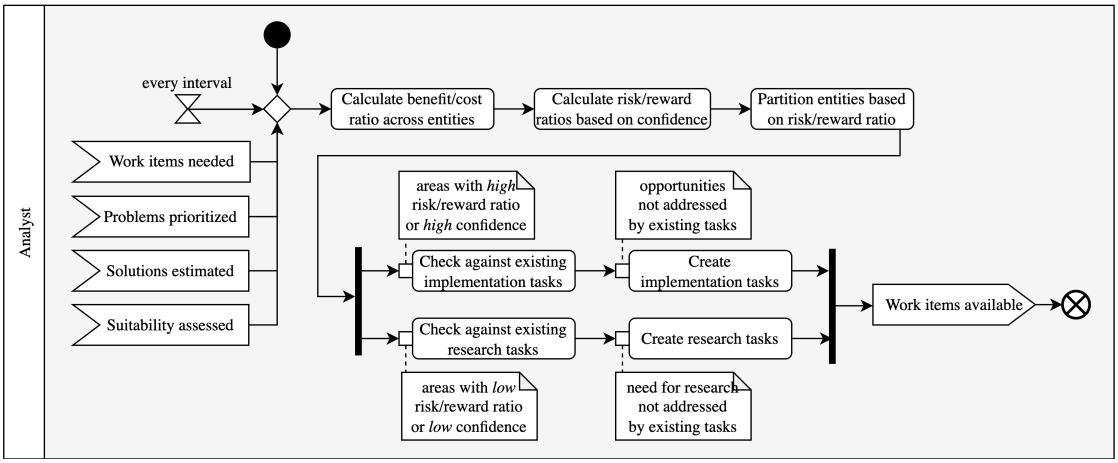


Figure 3.13: Process for continuously detecting need for research and opportunities for implementation.

This process may play out differently in practice depending on the maturity of the organization, the phase in the project lifecycle, or the specific instantiation of CORTEX.

The problem solving process attempts to maximize the benefit/cost ratio of the implemented solutions. This can be achieved by a number of ways: solving high-value problems with reasonable implementation effort, solving medium-value problems with especially cheap solutions, choosing more costly solutions that solve multiple problems at once, betting on high-risk but also high-reward implementation efforts to pay off. The way these decisions are made is shaped by the project parameters such as thresholds for minimum benefit and maximum cost as well as risk tolerance, optionally differentiated by the type of uncertainty causing it.

Overall, the process applies a *satisficing* heuristic and improves over time in several aspects: its knowledge about the problem and solution domain, its knowledge about sources and degree of uncertainty, the appropriate thresholds and tolerances in its specific context, the areas being explored for insights, the areas being exploited for business value.

The process is also *opportunistic* and can be tuned to either lazily or greedily react to new possibilities. Depending on the process parameters, it can eagerly experiment with new incoming problem descriptions as well as emerging technologies. This enables it to not only refine its course over time but also achieve breakthrough innovations. It can also replicate concepts like minimum viable features (MVFs) or minimum viable products (MVPs) as large-scale experiments with stopping conditions and pivoting to alternative courses of action [15]. Projects are encouraged to adapt this aspect to their priorities and re-evaluate it frequently as part of empirical process control [74, 22].

3.2.4 Decision Support Workflows

Sufficient knowledge about the business domain is crucial for identifying valuable features in product development [117, 96]. CORTEX aims to model or "map out" the problem and solution domain in order to direct research and implementation efforts towards the right areas at the right time. The decision support workflows aim to continuously explore and understand both the problem and solution domain. Its activities serve to coordinate the research activities by connecting the search for problems and solution, triggered by the problem solving process, to further investigation.

One caveat with this approach is the "paralysis by analysis" anti-pattern: The tendency to overanalyze the problem at hand or potential solution options [37, 8, 15]. As a first countermeasure, CORTEX allows organizations to decide how eagerly or lazily they want to approach problems and solutions, purposefully making the tradeoff between speed or thoroughness. To further counteract this issue, stopping conditions need to be established as part of the project parameters, e.g., the maximum depth to which a problem or solution is broken down, the maximum number of variants considered at each variation point, the maximum number of attempts to reduce uncertainty of a fact, etc. CORTEX does not model these cases of exception handling because they are highly dependent on the specific setup of each project and should therefore be addressed as part of implementing and tailoring each activity.

3.2.4.1 Problem Domain Exploration

CORTEX can be initialized flexibly depending on how the organization is set up, which timeframe the project has, in which phase of product development it starts, and how much is initially known about the problem and solution domain. The customer may report needs (e.g., as "problem statement" at the beginning of the project) and the R&D manager captures it as an initial starting point in the knowledge base. Alternatively, the problem solving process treats starts on the empty knowledge base, identifies this as a need for research and creates corresponding research tasks. Figure 3.14 shows the workflow for coordinating the activities that are elaborated on in the following paragraphs.

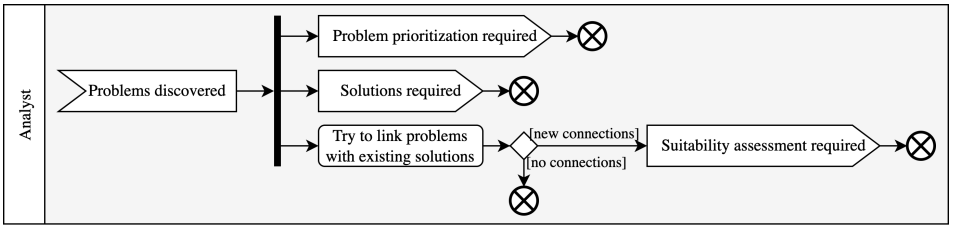


Figure 3.14: Workflow for coordinating discovery of entities in the problem domain with further investigation.

Problem Discovery Problems can be discovered either actively or passively. With an active approach, domain problems are elicited by researching and evaluating empirical

data. Passive discovery means reports from stakeholders, for example in the form of customer needs or user feedback. In both cases, further research is considered to validate the information before using it in decisions. For example, discovered problems might only interest a small number of stakeholders or require different solutions depending on which problem variant needs to be solved.

Problem Capture All information about the discovered problems is captured in the knowledge base. For each fact, all relevant factors of uncertainty are stored as well, such as confidence in the correctness of the problem description, variability of the problem, volatility of the problem context, etc. Information is stored as it becomes available and then updated continuously. This knowledge sharing enables distributed problem solving based on the blackboard pattern [116].

Problem Prioritization The process estimates the benefit of solving the problem while accounting for relevant factors of uncertainty. If an attempt at problem prioritization fails due data not being acquirable, another mechanism can be selected. Following lean principles, early validation aims to avoid wasteful implementation efforts until enough confidence is gained in the decisions. Furthermore, it counteracts the cognitive biases like the optimism bias whereby all problems are deemed valuable and the recency bias whereby the recently discovered problems are deemed more urgent.

Suitability assessment When a new problem is discovered, the process opportunistically tries to link it to existing solutions. This happens immediately after capturing the problem to avoid waiting for another loop. Vice versa, newly found solutions may be linked to the problem when they are captured. Whenever a connection is established, its suitability needs to be assessed to ensure there is actual problem/solution fit.

Solution Search The process searches for further solutions that might be suitable to the problem, as elaborated in the next section. To avoid "paralysis by analysis"[4], further solution search might be skipped if sufficient connections could previously be established.

3.2.4.2 Solution Domain Exploration

Similar to the view of the problem domain, the view of the solution domain can also be initialized depending on the context that CORTEX is applied in. The organization might be a startup focused on pursuing the possibilities of a new technology, the customer might bring along solution ideas, or domain experts might propose suitable approaches very early on. Alternatively, the problem solving process treats starts on the empty knowledge base, identifies this as a need for research and creates corresponding research tasks. Figure 3.15 shows the workflow for coordinating the activities that are elaborated on in the following paragraphs.

[4] B. Kane. *The Science of Analysis Paralysis: How Overthinking Kills Your Productivity & What You Can Do About It.* 2015. URL: `https://doist.com/blog/analysis-paralysis-productivity/`.

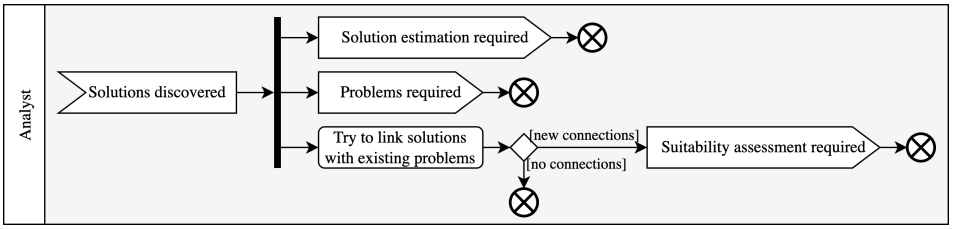


Figure 3.15: Workflow for coordinating discovery of entities in the solution domain with further investigation.

Solution Discovery Similar to problem discovery, there is a difference between actively searching for and passively receiving potential solutions. Active discovery means using research activities to explore the solution domain. Passively received solutions can either be stated by domain experts, stakeholders or other involved sources. In both cases, further research is considered to validate the information before using it in decisions. For example, discovered solutions might not actually fit the problem they were intended for or required technology might incur significantly higher costs than anticipated.

Solution Capture All information about the discovered solutions is captured in the knowledge base. For each fact, all relevant factors of uncertainty are stored as well, such as confidence in the correctness of the solution description, variability of the solution components, volatility of the solution technology, etc. Information is stored as it becomes available and then updated continuously. This knowledge sharing enables distributed problem solving based on the blackboard pattern [116].

Solution Estimation As the counterpart to problem priority, the process estimates the cost of implementing the solution while accounting for relevant factors of uncertainty. If an attempt at solution estimation fails due data not being acquirable, another mechanism can be selected. Following lean principles, early validation aims to avoid wasteful implementation efforts until enough confidence is gained in the decisions. Furthermore, it counteracts the sunk cost fallacy whereby a costly course of action would be continued irrationaly to not "lose" the previously invested effort.

Suitability assessment When a new solution is discovered, the process opportunistically tries to link it to existing problems. This happens immediately after capturing the solution to avoid waiting for another loop. Vice versa, newly found problems may be linked to the solution when they are captured. Whenever a connection is established, its suitability needs to be assessed to ensure there is actual problem/solution fit.

Problem Search The process searches for further problems to which the solution might be suitable, as elaborated in the previous section. To avoid "paralysis by analysis", further problem search might be skipped if sufficient connections could previously be established.

3.2.5 Continuous Execution Workflows

Data-driven product development collects and evaluates data to understand its domain and choose a successful course of action [27]. Instead of trying research everything beforehand, changes and unforeseen events are embraced as a chance to improve the existing knowledge base and cope with fast-changing, unpredictable environments [23, 10]. CSE tightly integrates implementation, verification, and delivery, creating the ability to continuously deploy new software increments to a target environment [7, 8, 9]. CORTEX ties into these processes, executing research and commissioning on-demand using asynchronous workflows driven by events. Progress about these workflows is reported equally by triggering events that can be consumed by the surrounding processes.

3.2.5.1 Demand-Driven Research

CORTEX integrates research into the continuous processes of CSE. Research activities are therefore executed *on-demand* for select areas or even specific entities in the problem and solution domain. Figures 3.16 to 3.18 visualize the workflows to discover and capture these entities (problems, solutions, connections) and assess the associated facts (the entities themselves, benefit, cost, suitability) to reduce the inherent uncertainty.

Discovering Entities For example, empirical methods like surveys or case studies are useful for building up initial knowledge about the problem domain. An important aspect here is the "solvability" of a problem or solution. If this is questionable, the entity itself is stored as a fact with a high degree of uncertainty. CRDM additionally encourages to break down entities into clusters or tree-like structures[5]. Once the project resolves the uncertainty inherent in these areas, the entities can be picked up by the problem solving process [61]. Appendix C.2.1 describes applicable strategies and methods for performing exploratory research activities.

Investigating Facts The first question after capturing a problem regards the problem's value: If the value of the problem is low, then the problem does not actually exist or is not worth solving [120]. Similarily, to be able to select solutions, they first have to be individually estimated according to defined criteria [121]. Appendix C.2.2 lists applicable techniques for this type of research. Research activities need to consider variability in the problem and solution space space. For example, a feature might be valued differently by different target audiences and it might also impact a different number of users in each audience. Similarly, a solution might incur different costs based on different techniques to implement it and it might also require other solution components in each case. Consequently, connections might only exist between specific variants of problems and solutions and each carry a varying degree of uncertainty.

[5] L. Lin. *McKinsey Issue Tree Example: Problem-solving and Decision-making.* 2011. URL: `https://www.slideshare.net/interviewcoach/mckinsey-issue-tree-example`.

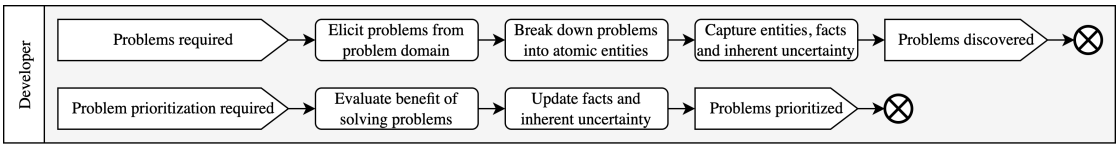


Figure 3.16: Workflow for discovering entities and investigating facts in the problem domain.

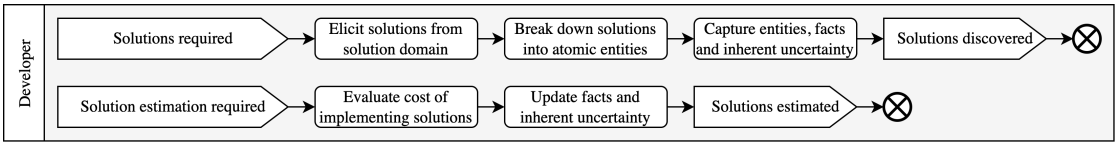


Figure 3.17: Workflow for discovering entities and investigating facts in the solution domain.

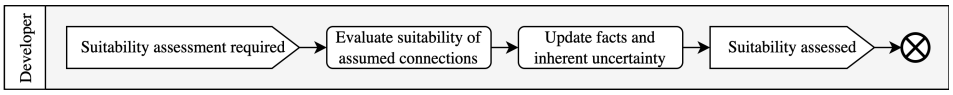


Figure 3.18: Workflow for discovering entities and investigating facts across problem and solution domain.

3.2.5.2 Progress Feedback

CORTEX tightly integrates with continuous experimentation and continuous delivery as defined by CSE. Continuous R&D presumes that the methodological and technological preconditions for continuous delivery, such as lean management and continuous integration, are in place. CRDM therefore simply defines *tasks* for conducting research or implementing a solution. Each task encompasses all necessary measures to integrate, verify, and deliver the research activity or implemented change. This notably does not imply that any one solution will be fully implemented with a single task. Instead, a solution can require an arbitrary number of implementation tasks, as apparent in the CRDM core model.

Since the tasks are buffered in a work item queue and workflows are executed asynchronously, both the problem solving process and decision support workflows need a feedback mechanism to be informed about the status of work items. Since CORTEX is event-driven, the feedback mechanism relies on events as well. In addition to the workflow-specific events that coordinate the execution of research and implementation, CORTEX defines low-level events to signal that work items are available, have been started, and have been finished. Figure 3.19 visualizes how this signaling mechanism ties into a development process for experimentation and delivery.

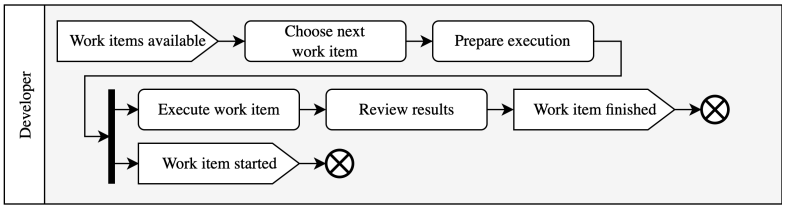


Figure 3.19: Workflow for providing feedback about the status of work items.

3.2.6 Process Tailoring

Project tailoring means customizing a process to a specific environment, set of constraints, or other project characteristics. It can be crucial for the successful adaption and execution of the process. We therefore identify areas of flexibility within the overall workflow as well as individual sub-workflows to allow for adequate tailoring. In these places, projects can and should define their own mechanisms and criteria to make full use of CORTEX according to their needs. However, projects should be careful to not alter the actual process in tailoring.

Problem/solution search frequency CORTEX is focused around opportunistic exploration of the problem and solution space of a project. Projects need to fine-tune how frequently they need to actively search for problems and solutions (section 3.2.3.2) or how much they rely on these items being proposed to them via stakeholder interaction (section 3.2.3.1).

Work item generation frequency Similar to search frequency, projects need to define how frequently they want to analyze the knowledge base in order to find new work items as part of the opportunistic search (section 3.2.3.2). This is crucial for proper integration of CORTEX with the development process. Projects should aim for a consistent "rhythm" across decision-making, research, and implementation.

Problem prioritization The CORTEX workflow specifies that problems should be prioritized for the opportunistic search to function correctly. This information is used to select high-value problems. However, CORTEX does not prescribe how to value problems as part of the research workflow (section 3.2.5.1). "Value" is an abstract term here expressing the gain that a project anticipates by solving a problem. Projects should define suitable metrics for describing that value along with a corresponding mechanism for estimating it. Appendix C.2.2 lists possible options for prioritizing items based on benefit and/or cost.

Solution estimation Estimation of solution cost is kept similarly abstract in the definition of the research workflow (section 3.2.5.1) but is also necessary for the opportunistic search. However, it is equally important for work item generation. Projects should define an appropriate mechanism to estimate and record solution cost. This depends on which criteria they need to capture in that estimation: complexity, effort, time, technical scope, degree of familiarity etc. The options listed in appendix C.2.2 apply here as well.

Research methodology Every activity in the research workflow (section 3.2.5.1) will be different depending on the workflow that triggers it, the specific context, the project's current state, the questions or hypotheses to be investigated, and potentially many more factors. Projects therefore not only need to define their research methodology in general, i.e., which method they apply for which type of research, but also decide how to conduct research on a case-by-case basis.

Research limits Equally important as the research methodology is a stopping condition for the research as part of the opportunistic search (section 3.2.3.2). Basically, projects need to define when to stop a research activity, e.g., searching for a solution, and instead focus on more urgent tasks. They can also decide to strictly limit ad-hoc research and rely on the subsequent triggering of additional research. Such limits could be a time box in which any activity is expected to finish, how many alternatives to uncover in a solution search, or how many samples to collect in an experiment, and how strictly to evaluate results.

Knowledge base analysis Depending on the structure of the project's problem and solution space, we anticipate that different search algorithms (e.g., depth-first vs. breadth-first) are needed to uncover to hot/cold spots appropriately when analyzing the knowledge base (section 3.2.3.2).

Risk tolerance CORTEX aims for projects to consciously make decisions about how much risk to take on when deciding on their next work items (section 3.2.3.2). In doing so, projects apply a risk tolerance that determines whether a given area in the knowledge base is viewed as an opportunity for implementation and, conversely, where need for additional research is identified. This aspect of decision-making is deemed essential for applying CORTEX and therefore should be defined explicitly and carefully. This activity is supported by research where risk tolerance can serve as a guideline for how extensive the research needs to be and how strictly it needs to be evaluated (see "Research limits" above).

Work item management Finally, projects might use any type of system and methods to capture, prioritize, and track their work items. CORTEX does not prescribe anything here, work item generation (section 3.2.5.2) should be adapted to the preferred setup to allow for a seamless integration. The goal here is to propagate knowledge about a work item's priority and to feed back information about a work item's progress and outcome.

3.3 Validation of CORTEX Integrity

The design cycle for CORTEX concludes with validation of the designed treatment. This validation is divided into two parts, the first of which investigates knowledge question 5: How does CORTEX behave in CSE projects? Wieringa [63] proposes requirements satisfaction questions that investigate both the behavior and performance of the treatment in the anticipated context. This is analogous to model validation in OOSE, specifically validation of the CORTEX Analysis Model with focus on its dynamic model. This validation therefore covers the following research questions:

1. Does the design of CORTEX cover all relevant elements of CRDM?
2. Does the stimulus-response behavior of CORTEX satisfy functional requirements?
3. Does the performance of CORTEX satisfy nonfunctional requirements?

Wieringa [63] defines validation of requirements satisfaction in the following sense: "If the requirements for the treatment are specified and justified, then we can we validate a treatment by showing that it satisfies its requirements."[6] In this case, requirements refer to both the functional and nonfunctional requirements from the Continuous R&D ecosystem. Figure 3.20 visualizes the use of simulation models as analogical representations of process and projects. Section 3.3.1 describes the single-case mechanism experiment that validates the integrity of the CORTEX process model using a software simulation. Section 3.3.2 assesses the completeness, consistency, and correctness of CORTEX behavior according to the previously established requirements. Section 3.3.3 elaborates on the satisfaction of nonfunctional requirements by CORTEX performance.

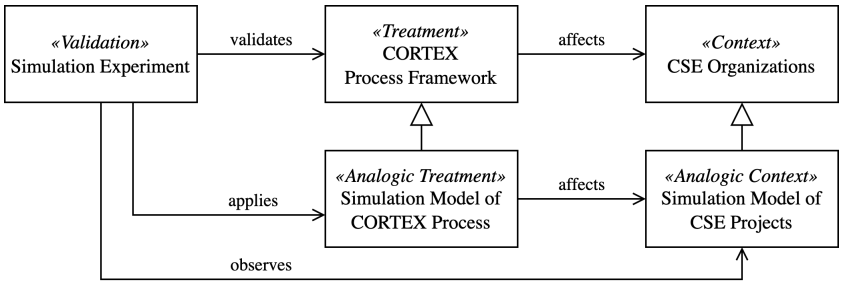


Figure 3.20: Validation of CORTEX using simulation models of both process and projects.

[6] Despite the term "validation" in the context of design science and OOSE, this particular activity is closer to *verification* which tests objectively whether a subject fulfills the properties specified in requirements. Formally it is defined as "the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition." [107]

3.3.1 Simulation Methodology

A simulation model serves to investigate the research questions in *single-case mechanism experiment*, described by Wieringa [63] as follows: "In single-case mechanism experiments, individual cases are experimented with in order to learn which phenomena can be produced by which mechanisms. The cases may be social systems or technical systems, or models of these systems. [...] We often speak of *testing* a technical prototype or of *simulating* a sociotechnical system."

Simulation experiments execute a simulation model with specific parameters, then analyzes collected data. Their primary advantage lies in investigating expensive, unreal, dangerous, or unethical aspects of the research project. Additionally, they can simplify systems and processes through abstraction to focus on relevant aspects and control the environment. The simulation setup can also collect data as required for analysis in terms of time, place, precision, and quality. Conducting a simulation experiment involves the following activities [122]:

1. Modeling: Reflect or abstract reality as appropriate.
2. Calibration: Tune parameters to reproduce known result.
3. Verification: Ensure correct implementation of concept.
4. Validation: Ensure accurate representation of reality.
5. Execution: Run model with set of parameters.
6. Evaluation: Collect data and interpret results.

The simulation experiment is divided into two stages: Pre-development of the simulation model, followed by a re-implementation of the model for actual execution. Pre-development is conducted in commercial simulation software AnyLogic[7] because of its powerful capabilities for multiple modeling paradigms: Discrete event simulation (DES) tracks individual state changes in an event-based process; system dynamics modeling (SDM) helps to understand nonlinear behavior of complex systems using stocks and flows; agent-based modeling (ABM) allows observing interactions between autonomous parts of a system and emergent behavior. Figures 3.21 and 3.22 show the exploration of single- and multi-paradigm models.

Despite the extensibility of AnyLogic, final results are obtained from a reimplementation of the simulation model in Swift[8], a type-safe language that supports object-oriented and functional programming. This allows for a high level of confidence that the simulation model is actually replicating CORTEX as intended and not behaving differently in unexpected or undetected ways. While SDM is not required, ABM seems suitable for modeling the real world. However, modeling individual actors can actually be replaced by introducing stochastic parameters. CORTEX validation therefore uses a stochastic DES model. Appendix C.3 contains excerpts from the Swift codebase of the simulation.

[7] `https://www.anylogic.com/`
[8] `https://www.swift.org/`

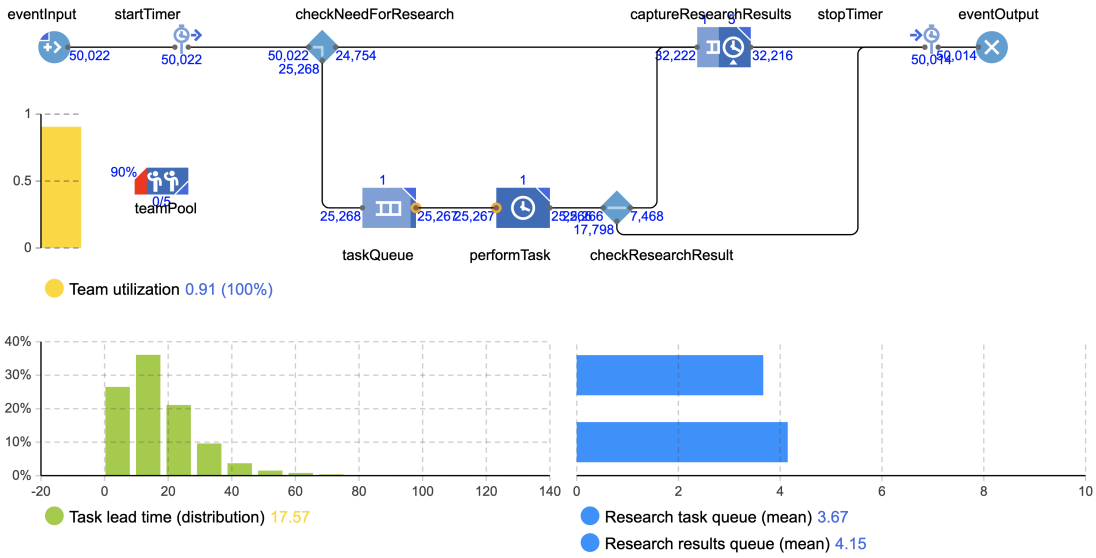


Figure 3.21: Single-paradigm model of task processing in AnyLogic, using DES with multiple queues and conditional switches.

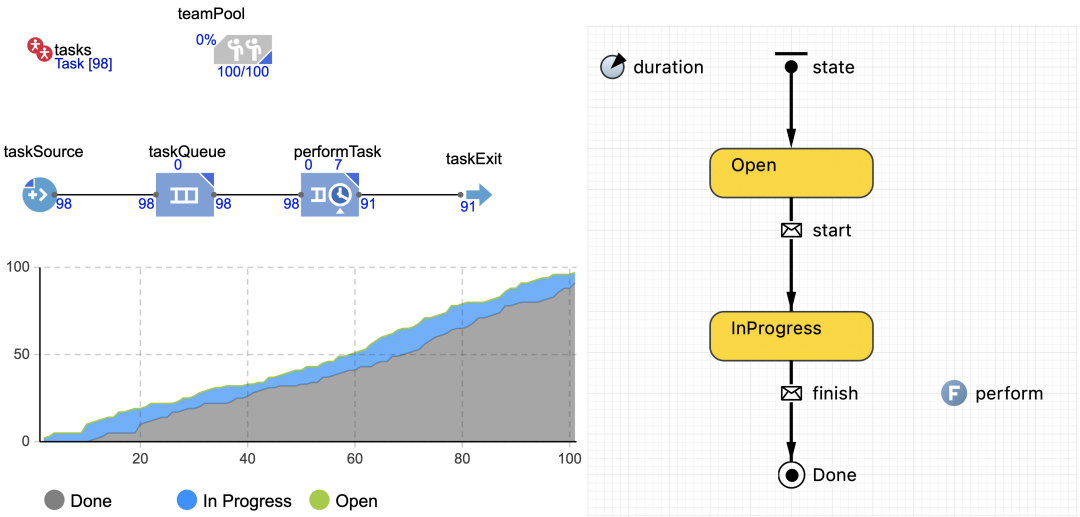


Figure 3.22: Multi-paradigm model of task processing in AnyLogic, combining DES with ABM and a custom state machine.

3.3.2 Behavioral Integrity

The simulation model of CORTEX was able to cover the entire CRDM model, demonstrating *completeness* regarding its foundation. Using type-safe implementation of the simulation model demonstrated that CORTEX was *consistent* regarding the interconnections between its overarching problem solving process and the individual workflows. *Correctness* of CORTEX with respect to its requirements was ensured using both compile-time and run-time checks of the simulation model. The following paragraphs list how functional requirements were verified by utilizing the type system and validated by applying automated tests as well as analyzing data from executing the simulation model.

FR 1: Capture problems and solutions.

- ✓ Verified using type system (all entities and knowledge base)
- ✓ Validated using unit tests (adding and replacing entities)
- ✓ Validated using simulation results (build-up of knowledge base)

FR 2: Estimate benefit, cost, uncertainty of entities.

- ✓ Verified using type system (entity attributes, value types)
- ✓ Validated using unit tests (estimation, confidence calculation)
- ✓ Validated using simulation results (reduction of uncertainty)

FR 3: Discover and seize opportunities for problem-solving.

- ✓ Verified using type system (decision algorithm via function composition)
- ✓ Validated using unit tests (benefit/cost ratio, minimum ROI, optimal solution)
- ✓ Validated using simulation results (recurring implementation bursts)
- × No check for globally optimal decisions across all possibilities (case study required)

FR 4: Validate known problems, solutions, and connections.

- ✓ Verified using type system (entity-specific research tasks, optional result)
- ✓ Validated using unit tests (updates to knowledge base, handling missing result)
- ✓ Validated using simulation results (alternating research and development)
- × Simulation model abstracts from specific reason for missing result (irrelevant here)

FR 5: Identify need for further research.

- ✓ Verified using type system (uncertainty as attribute, confidence as value type)
- ✓ Validated using unit tests (confidence calculation, research when confidence is low)
- ✓ Validated using simulation results (continuous generation of research tasks)

FR 6: Identify potential for implementation.

- ✓ Verified using type system (benefit and cost as value types)
- ✓ Validated using unit tests (calculation of ROI, check for existing tasks)
- ✓ Validated using simulation results (continuous generation of implementation tasks)

FR 7: Support experimentation using suitable research methods.

- ✓ Verified using type system (entity-specific research tasks, optional result)
- ✓ Validated using unit tests (handling indeterminate duration and missing result)
- × Simulation model otherwise abstracts from research method (irrelevant here)

FR 8: Support experiments in laboratory and real world.

- × Simulation model abstracts from research environment (case study required)

FR 9: Guide use of research results in decision-making.

- ✓ Verified using type system (functions requiring knowledge base)
- ✓ Validated using unit tests (updates to knowledge, use of up-to-date knowledge)

FR 10: Facilitate learning from research.

- ✓ Verified using type system (iterations carry evolving knowledge base)
- ✓ Validated using unit tests (updates to knowledge base, transfer between iterations)

3.3.3 Behavioral Characteristics

We investigated whether performance of CORTEX satisfies nonfunctional requirements (see section 2.1.3). The following paragraphs summarize the observed performance of CORTEX in the relevant aspects.

Usability and Intuitiveness Usability was out of scope for the simulation experiment and simulated projects were simply able to execute CORTEX according to the specification. Similar to usability, the simulation is not suitable for validating the intuitiveness of applying the model. A positive indication could be that CORTEX was straight-forward to implement, "teaching" it to the machine compiler and runtime instead of real projects. However, these requirements should be validated properly and without bias in a case study under real-world conditions.

Performance The simulation successfully explored the the problem and solution domain, building up knowledge and reducing uncertainty. Whenever sufficient knowledge was available, CORTEX recognized the opportunities by calculating the expected value of the return on investment of implementation decisions (see the formula presented in section 3.2.3.2). Figure 3.23 shows how research leads implementation: Internally, the process builds up knowledge about entities and subsequently reduces uncertainty about them, leading to the fluctuating percentage of properly estimated entities. Implementation decisions are made only when sufficient knowledge with sufficient confidence is available, visible by the increase of estimated entities preceding the increase of solved problems. This coupling causes the percentage of solved problems to reflect the fluctuations in "decision-worthy" knowledge, decreasing as new problems are discovered and increasing

as they are solved. Redundant activities were successfully avoided both in research and implementation by checking for already existing tasks with identical goals.

Figure 3.23: Progress of problem solving in CORTEX simulation; research produces estimated entities, subsequent implementation produces solved problems.

Scalability The simulation ran stable at different scales of each simulation parameter. This was validated by parameterizing work capacity, occurrence of problems and solutions, and underlying uncertainty. Figure 3.23 visualizes task processing in the simulation. The graph shows how the process starts (with problem search) and as the first tasks are picked up by for processing, the work item queue is filled. Processing then stabilizes at a rate that is proportional to the resources available in the project.

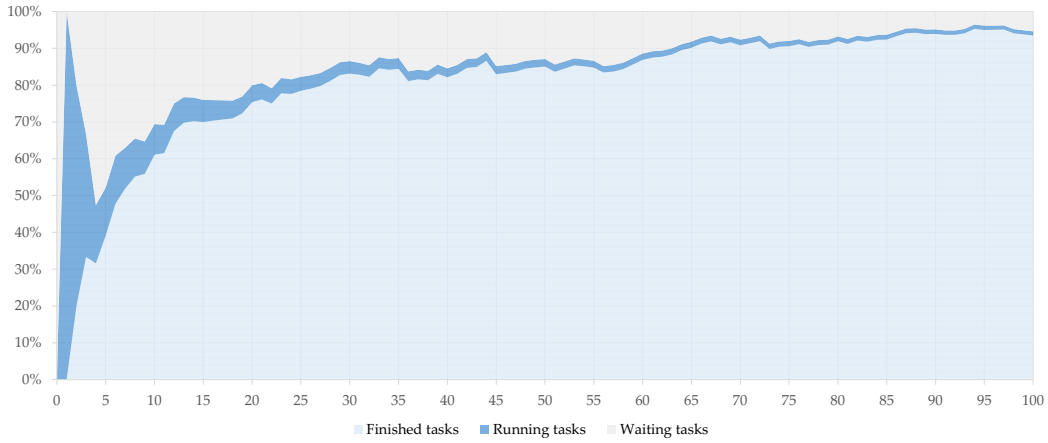


Figure 3.24: Progress of task state in simulation; initial bootstrapping phase is followed by constant utilization of available capacity.

Supportiveness and Customizability The fact that the model was able to abstract from specific methodologies for development and research indicates that CORTEX has no strong coupling to these aspects. To simulate different project phases, the simulation modeled phases with no knowledge, quickly building up knowledge, and slowing down generation of new knowledge. Figure 3.25 visualizes the distribution of task types over time and shows the different phases throughout the project. Figure 3.26 visualizes the distribution of effort by scaling implementation to a realistic ratio. Projects can influence the distribution and by adjusting their tolerance for uncertainty as well as minimum ROI of implementation activities. These requirements should be further validated in a case study under real-world conditions.

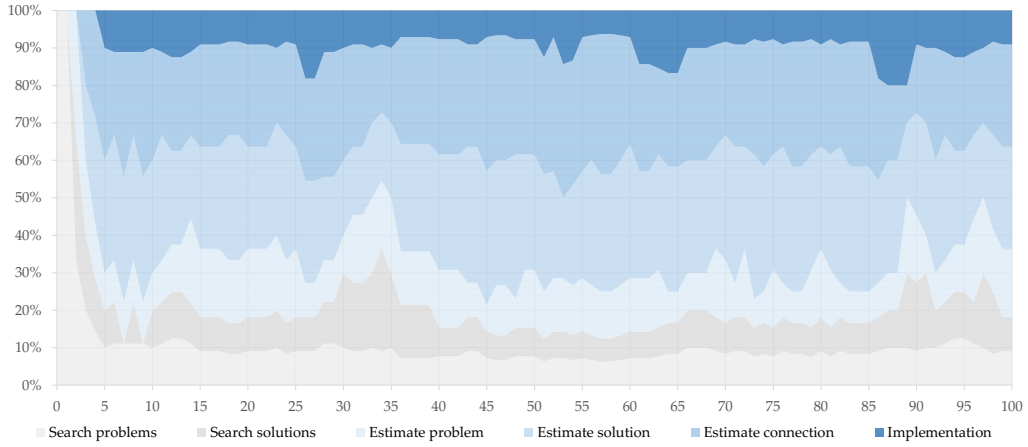


Figure 3.25: Progress of task distribution in simulation; bursts of research alternate with bursts of implementation.

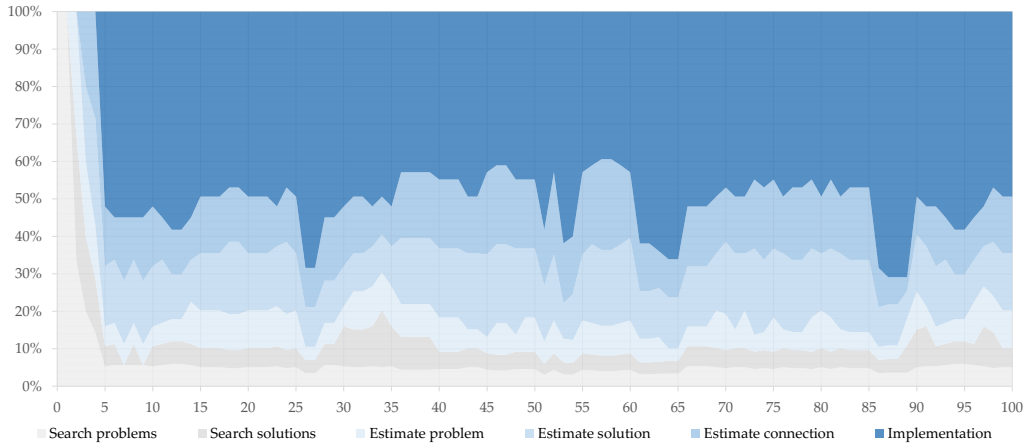


Figure 3.26: Progress of effort distribution in simulation; implementation scaled by real-world ratio of feature development to research.

3.4 Validation of CORTEX Effects

Following the validation of requirements satisfaction, CORTEX is further validated regarding its effects in the real-world problem context. This second validation activity investigates knowledge question 6: How does CORTEX impact decision-making in CSE projects? It therefore also concludes prediction goal 1: Predict the impact of using CORTEX in CSE projects. For this type of validation, Wieringa [63] proposes research questions that ask whether applying the artifacts in the context produces effects, how the artifacts responds to stimuli and which performance it shows in the context. Applied to CORTEX, this entails the following research questions:

1. Does CORTEX address relevant types of decision problems and support decision making?

2. Are organizations able to solve problems more effectively using CORTEX?

3. Do the additional research activities negatively impact the timeliness of development?

4. Does the research-driven approach affect the quality of the delivered solutions?

In terms of OOSE, this is again analogous to model validation of the CORTEX Analysis Model. However, the focus is no longer on the correctness of the design with regards to its requirements. Instead, validation focuses on the effects, or also complications, that can be anticipated in the problem context. Both primary effects on decision-making and side-effects on quality and efficiency are to be considered when assessing the suitability of CORTEX for the goals of Continuous R&D.[9] Figure 3.27 visualizes how the validation uses analogical representations of both artifact and context and then draw conclusions using analogic inference.

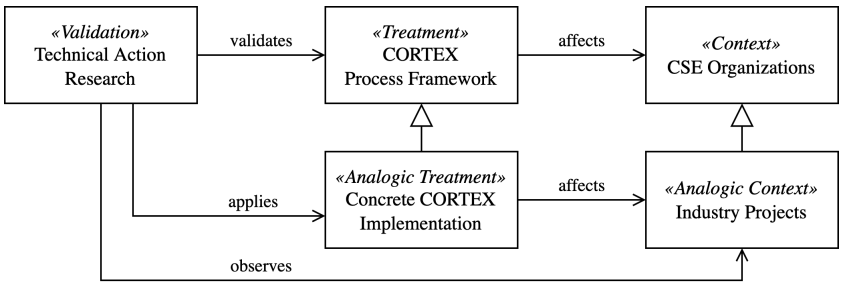


Figure 3.27: Validation of CORTEX using a concrete implementation in industry projects.

[9] In contrast to the previous validation activity, this activity is true to the idea of validation and tries to find objective evidence that a subject meets the prerequisites for a specific application or use. Formally it is defined as "the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers." [107]

Section 3.4.1 describes the case study that validates CORTEX effects using technical action research. Section 3.4.2 assesses the effects of applying CORTEX on decision-making, specifically decision effectiveness and error detection. Section 3.4.3 describes side-effects of CORTEX on product quality as well as execution efficiency. Section 3.4.4 presents additional anecdotal evidence from CSE projects that strengthen the results of the case study.

3.4.1 Case Study Methodology

CORTEX is field-tested using a case study in a business environment which is then evaluated using an end user survey [123, 124]. The goal is to assess how the problem solving process and its workflows perform in a real-world context. The case study observes whether reducing uncertainty supports decision making and how it impacts the project's success factors. Data is collected from two main sources: Insights collected during the project case study, and results from the end user survey. Additional anecdotal evidence is used to cross-validate the results. Findings should validate the suitability of the process model for the context of Continuous R&D and potentially reveal potential weaknesses or gaps.

To validate the impact of CORTEX on project success, validation needs to define how to measure it. Literature on empirical software engineering contains numerous descriptions of project constraints and success factors. A prominent example is the "iron triangle" of scope, time, and cost with quality as an additional property that emerges over time [45]. This model can be augmented with other factors such as system quality, service quality, user satisfaction, learning effects, impact on stakeholders, and organizational benefits [125]. Bloomberg [126] proposes changing the triangle dimensions to agility, quality, and time. Ralph and Kelly [127] try to summarize factors in a "software engineering success framework": Projects produce artifacts which perform in markets; projects exhibit efficiency, artifacts exhibit quality, and markets exhibit performance. Validation therefore differentiates *effectiveness*, *efficiency*, and *quality* as dimensions of project success, which corresponds to the latter three research questions (in that order).

Regarding decision support, the research hypothesis is that by applying Continuous R&D, projects will be able to make better decisions earlier in time. An indicator of success is how many decision problems remain unresolved and how many strategic changes ("pivots") occur in product development. Regarding effectiveness, efficiency, and quality, the research hypothesis is that CORTEX will improve effectiveness while not impacting efficiency and quality – at least not negatively. This is based on the assumption that, despite additional effort being spent on research, time is saved by reducing waste in the form of bad decisions and better decisions lead to better quality.

3.4.1.1 Case Study

A multi-project case study validates the effects of CORTEX on effectiveness, quality, and efficiency [123, 124]. Validation is conducted in the field to investigate the effects of a real-world implementation. It represents a series of *single-case mechanism experiments*

as described by Wieringa [63]: "[T]he researcher studies individual cases, just as in observational cases studies, to investigate how phenomena in the case are produced by the architecture of the case. But in single-case mechanism experiments, the researcher intervenes, i.e., experiments, with the case. [...] Single-case mechanism experiments are useful for validating new technology, for evaluating implementations, and for investigating problems in the field."

This validation is classified as *technical action research* (TAR) since it not only answers a knowledge question but also helps a client: the real-world project. The goal is to determine how CORTEX performs under real-world conditions and which effects can be observed over an extended time period [104]. Since in TAR researchers have access to the project environment and stakeholders, additional insights can be generated through stakeholder observation. Specifically, the case study collects data from issue trackers for independent analysis as well as triangulation of survey results [128, 122]. Even though the case study is not set up as a controlled experiment, data is collected from similar projects in each project's environment to allow controlling for external influences and seasonal effects.

The case study is comprised of 10 projects with a total of ∼70 developers. The cumulative observation period across all projects spans 120 weeks, amounting to ∼4,000 person days[10] of practical application of the process framework. Figures 3.28 and 3.29 show the distribution of developer count and observation period across projects. Selected projects exhibit a high degree of complexity in company environment, organization, and product development. In large organizations, selected projects are part of an "intrapreneurship environment"[11] to grant freedom to the research. Selected departments are measured by their impact on real-world metrics: creation of business value, adherence to delivery date, quality grade of developed software. This environment allows to both introduce the CORTEX process framework in a controlled fashon and examine its effects on the project under realistic conditions. Appendix C.4.1 contains examples of artifacts produced by a project in a large-scale organization.

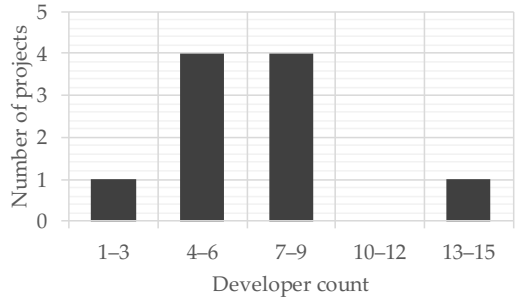


Figure 3.28: Distribution of developer count across projects in case study.

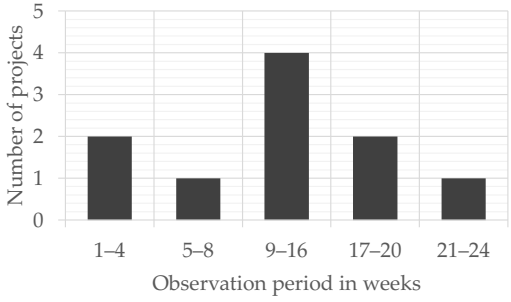


Figure 3.29: Distribution of evaluation period across projects in case study.

[10] The sum product of developers and observation days per project: $\sum_{i=1}^{10} developers_i \cdot days_i$

[11] An environment within a large organization that is granted additional freedoms to be able to behave in an entrepreneurial fashion as if it was a separate entity in the market.

3.4.1.2 User Survey

In addition to observational data from the execution of the case study, a qualitative user survey collects feedback from stakeholders to assess the outcome of product development. The survey uses a questionnaire to save time compared to individual interviews. Stakeholders that are asked to fill out the survey range from users over managers to technical experts. In organizations with large numbers of stakeholders, representative samples are chosen from each group according to their role or department. This yields an average participation rate of 21 % across thousands of addressable stakeholders. Additionally, stakeholders are granted sufficient time to evaluate the project solution before being surveyed and the findings are triangulated with observations from the case study to increase reliability [104, 128].

The survey structured questions into five categories to measure the effects of CORTEX according to the initially formulated research questions. The survey presented 2-5 questions in each category, asking for the criteria to be rated on a numerical Likert scale [71] that ranged, based on the German system of school grades, from 1 ("excellent") to 6 ("deficient"). It additionally allowed respondents to give textual feedback on the process and results as well as suggestions for improvements. Table 3.4 shows the line of questioning within each category. The specific questions that a project is asked may vary depending on its context, i.e., constraints, problem and solution domain, and stakeholders. Appendix C.4.2 contains examples of results produced by the user survey in a project in a large-scale organization.

Survey Category	**Line of Questioning**
Innovation/improvement	Does the product represent a novel or improved solution?
	Have problem statements been properly addressed?
	How much does the solution improve on stakeholder needs?
Functionality/robustness	How well do features match the respective problems?
	Have requirements been captured completely and correctly?
	Is the solution quality acceptable, i.e., error-free and stable?
Usability/performance	Can the solution be learned and adopted in an intuitive way?
	Does its use feel easy and comfortable or complicated?
	Does the performance match requirements and expectations?
User interface design	Is information presented correctly and understandably?
	Can functionality be discovered and used without support?
Overall satisfaction	Does the solution match the expectation from development?
	Is the solution worth recommending to similar stakeholders?

Table 3.4: Line of questioning within each category of the user survey.

3.4.2 Effects on Decision-Making

The end user survey collected qualitative data on the delivered solutions from stakeholders, ranging from users over managers to technical experts. This data served as a proxy to assess whether CORTEX makes projects more effective at problem solving. Two criteria were employed to measure the quality of the decisions based on their outcome: First, by properly exploring the problem and solution domain, the solution should match the problem and solve it well according to the stakeholder's needs. This is captured as **problem/solution fit** [35]. Second, by using the existing state of the art as input to the knowledge base, the implemented solution should be at least as good as comparable solutions on the market. This is captured as **innovativeness** [10].

Figures 3.30 and 3.31 show the rating of problem/solution fit and innovativeness, respectively. Both distributions are skewed towards 1 ("excellent") and 2 ("good") with a mean rating of 1.85 on problem/solution fit and 1.75 on innovativeness. The second peak at 4 ("adequate") in both dimensions was investigated and revealed to be based on stakeholders with additional needs. These needs hadn't been discovered or addressed yet, leading the respondents to rate the product as unsatisfactory ("satisfactory" would have been a score of 3).

Feedback furthermore suggests the following benefits of the research-driven problem solving process [124]:

- Structured overview of facts and areas of uncertainty,
- conscious evaluation of decision components such as priority, effort, or alternative,
- easier decisions once comfortable level of confidence was reached,
- faster discovery of mistakes and less changes or pivots,
- implicit generation and documentation of rationale of these decisions.

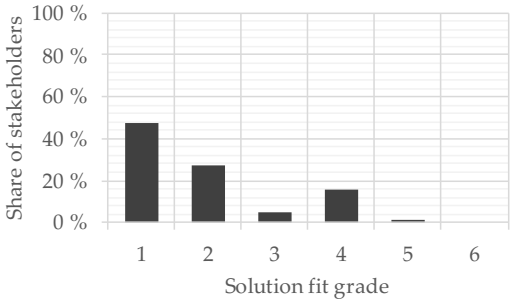


Figure 3.30: Distribution of rating of problem/solution fit across all survey respondents.

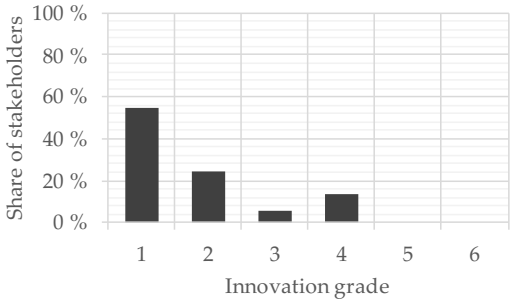


Figure 3.31: Distribution of rating of product innovativeness across all survey respondents.

3.4.3 Side Effects on Efficiency and Quality

The secondary concern of this validation is whether CORTEX negatively impacts execution efficiency or output quality of product development. This concern is understandable since CORTEX involves additional effort for designing, conducting, and evaluating research activity. However, the research hypothesis in this regard is that the additional effort is outweighed by the time and trouble saved through making better decisions earlier in time. Timely recognition of wrong assumptions should result in fewer pivots, changes, or reverts and therefore more time being spent on proper execution and refinement. Therefore, CORTEX is not expected to impact execution efficiency or product quality negatively.

Using the data collected from observations during the case study as well as form the und user survey, the validation finds no negative effect on efficiency or quality. Instead, throughput stays constant regardless of whether or not CORTEX is applied, suggesting that although customer involvement and experimentation need time, less effort is directed at wasteful activities. Feedback on product quality indicates that the problem solving process resulted in an improved use of time and attention, allowing nonfunctional requirements such as usability and sustainability to get addressed as well. Based on how serious this concern was perceived by projects, these are important insights that should be strengthened further with future research.

3.4.3.1 Effects on Execution Efficiency

Observation of projects before, during, and after the evaluation period collected data on **lead time** and **cycle time** of work items as important metrics for the efficiency of development projects. Both metrics describe the duration of executing a work item, lead time measures the duration from creation until completion while cycle time measures the duration from start to completion. Lead time therefore is an indicator for the ability of the organization to react to input and deliver corresponding output. Cycle time is an indicator for the speed at which output is generated and thereby the throughput of the development process [7, 8, 63].

To ensure both metrics reflect effects of the process model on efficiency, other project parameters were controlled. Additionally, data was collected from control projects in the same environment to control for external influences. Work items were differentiated into research tasks and implementation tasks and analyzed separately. The collected data do not indicate any negative effect on efficiency:

- Start of development: Lead times with/without CORTEX showed no significant differences in time spent in the backlog before start of development.
- Delivery of functionality: Cycle time with/without CORTEX showed no significant shift, neither within the same project nor compared to controls.
- Research vs. implementation: Cycle time did not vary significantly between research and development tasks, neither within the same project nor compared to controls.

As visual representation serves the "control chart" report from Atlassian Jira that visualizes cycle time along with its total average, rolling average, standard deviation, and significant outliers[12]. Figure 3.32 shows the cycle time of a representative project remaining stable over 28 weeks including an 18 week validation period of CORTEX. Figure 3.33 shows the cycle time of another representative project with a stable timeline. The increase around Christmas and New Year can be corrected as a seasonal effect using data from control projects, as shown in fig. 3.34.

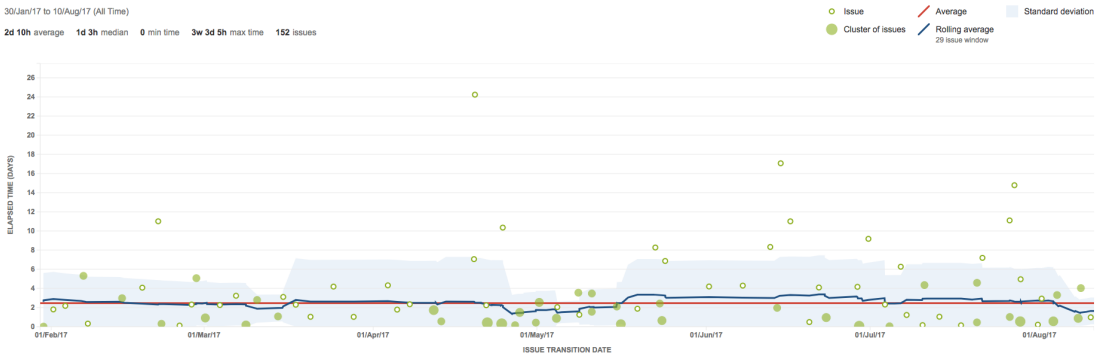


Figure 3.32: Control chart from representative project showing progression of cycle time (chart from Atlassian Jira).

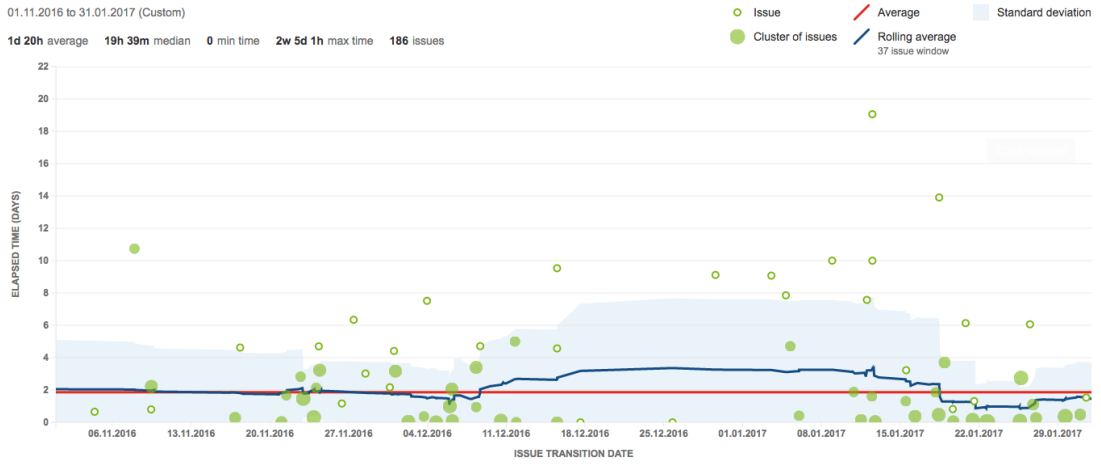


Figure 3.33: Control chart from representative project showing progression of cycle time (chart from Atlassian Jira).

[12]Similar to outliers in development time, some research activities took longer than expected to collect the required amount of data.

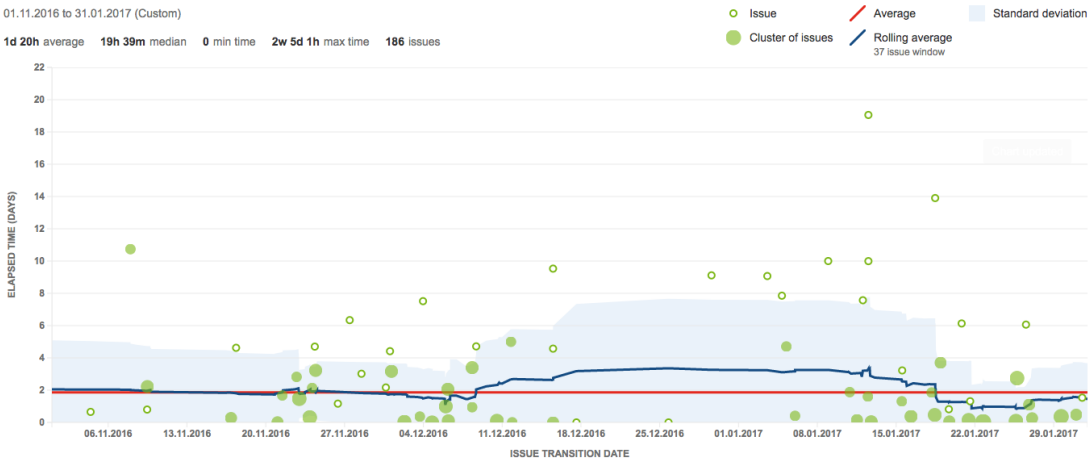


Figure 3.34: Control chart from control project showing progression of cycle time (chart from Atlassian Jira).

3.4.3.2 Effects on Product Quality

The end user survey collected qualitative data on product quality from stakeholders, ranging from users over managers to technical experts. Quality rating covered criteria such as ease of use, intuitiveness, performance, and robustness. Results were aggregated into two categories: **Technical quality** based on performance, stability, and maintainability; and **usability** based on visualization, intuitiveness, and ease of use.

Figures 3.35 and 3.36 show the rating of usability and technical quality, respectively. Both distributions are skewed towards 1 ("excellent") and 2 ("good") with a mean rating of 1.85 on usability and 1.81 on technical quality. Compared to results of the *Annual State of Agile Report*[13] where 25 % of respondents reported no improvement regarding software quality and 36 % reported no improvement regarding maintainability, this result suggests an above-average output.

This impression is further strengthened by stakeholder feedback: Giving stakeholders the option to provide an overall score of satisfaction with the project result yielded a mean grade of 1.66, which is even better than the average of the means of the other categories (1.83). Additionally, 94 % of the survey participants would recommend the developed product to others. This indicates that projects using CORTEX are able to produce a useful *and* well-implemented product.

[13] VersionOne Inc. *11th Annual State of Agile Report*. 2017. URL: `https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2`, VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`, VersionOne Inc. *13th Annual State of Agile Report*. 2019. URL: `https://explore.versionone.com/state-of-agile/13th-annual-state-of-agile-report`.

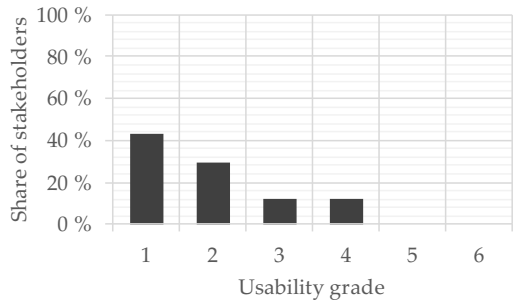


Figure 3.35: Distribution of rating of usability criteria across all survey respondents.

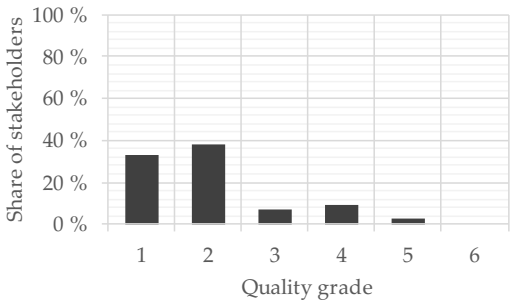


Figure 3.36: Distribution of rating of technical quality criteria across all survey respondents.

3.4.4 Additional Anecdotal Evidence

Additional anecdotal evidence was collected from observations as well as feedback from users and stakeholders and is used to cross-validate the results. The following notes were taken on the practical application of CORTEX and its components in the context of different case study projects. Some provide further insight into the implications of applying CORTEX in practice, while others strengthen the results of the previous data analysis. Table 3.5 lists which concrete mechanisms were employed by projects (see appendix C.2 for details).

Development Process CORTEX could be integrated regardless of the development process and the corresponding toolchain.

Research Activities Research workflow events were triggered quite often in early stages since assumptions need to be tested, later stabilized at a lower level. Ratio of development to research was individual to each project depending on their problem and solution domain. Research did not always yield results, e.g., project was unable to elicit proper or viable solutions due to a topic's complexity. Negative outcomes (errors, unusable results, timeouts) are currently only modeled superficially, process model needs extension.

Problem Search Projects received many problems passively through proposals by stakeholders, but still made sure to also elicit and validate domain problems on their own. A large project surveyed ∼400 out of ∼1.2k known stakeholders, refined the results with interviews, and analyzed ∼2.8k error logs.

Solution Search Projects often passively received some solution proposals from stakeholders, but still focused on their own technical knowledge and solution estimations to decide.

Problem Prioritization A project used RICE in combination with the Kano model to use both quantitative and qualitative information. Prioritization of domain problems was very constant and required little adjustment (hypothesis: because of extensive research).

Solution Estimation A project changed their prioritization model from RICE to RICC to move away from unreliably effort estimation. Projects often had to adjust the estimate of solutions, e.g., due to unpredicted effort or new insights from development.

Suitability Assessment Project used the Kano model to rank different combinations of problem/solution variants, effectively measuring the desirability of the resulting feature.

Effects on Decision-Making Projects rarely had to perform pivots in their course or radical changes to their approach during the case study. This substantiates the hypothesis that CORTEX enables them to make better decisions earlier in time and working with increased effectiveness. This would mean that the originally unmanageable risk posed by inherent uncertainty from different sources was reduced or mitigated. Reportedly, projects recognized misconceptions early on through exploration and thereby understanding of their domain and constantly testing their assumptions. Empirical evidence is often trumped by other factors such as urgency, taste, or politics, distorting the perceived value of research efforts.

Effects on Product Quality Product design received good ratings, serving as an example for the benefit of employing techniques like prototyping and user testing in the right amount at the right time. Some projects collect further objectives measures on product quality such as rating of code quality and incident resolution times. These criteria could be used in future research in addition to the measures on performance, stability, and maintainability. A project delivered solutions to groups of 100 stakeholders each and collected feedback directly.

Effects on Execution Efficiency Prototyping in the case study projects took anywhere from a day to two weeks depending on how many features the prototype contained and what fidelity was pursued. Some projects have large user counts, from hundreds to thousands of stakeholders with the potential of serving as test subjects and giving feedback, which in itself poses both an opportunity and a challenge. Projects with the ability to draw comparisons in their environment reinforced the hypothesis that improved decisions save as much time as is spent on research. Some projects report that at least the same amount of functionality was released within the same time frame of a normal release cycle. Furthermore, experimental efforts were conducted parallel to other activities, so standard development iterations were generally not slowed down.

Category	Mechanisms
Development process	Scrum, Kanban, combination of Scrum and Kanban ("Scrumban"), self-defined process based on agile/lean principles
Research activities	Exploratory strategy, explanatory strategy, quantitative methods, qualitative methods
Feedback collection	Scrum sprint reviews, product/feature launch presentations, user testing, support messages, user analytics
Problem search	Market research, stakeholder surveys, stakeholder interviews, expert interviews, log analysis, error analysis
Solution search	Prototyping, expert interviews, online research, benchmarks, user testing
Problem prioritization	RICE score, RICC score, Eisenhower matrix, Kano model, stakeholder voting (top 10), cumulative user voting
Solution estimation	Numerical sizing (person days), nominal sizing (story points, t-shirt sizes)
Suitability assessment	Prototyping, expert interviews, automated tests (behavior-driven development), Kano model

Table 3.5: List of mechanisms used by projects for applying CORTEX in their context.

4 RADAR: A Decision Support System for Continuous R&D

The third and last design cycle of this dissertation pursues artifact design goal 2 as well as prediction goal 2: Designing a decision support system for CORTEX based on CRDM and predicting the benefits of using the system to support the application of CORTEX. The result is the RADAR decision support system (Risk-Aware Development based on Agility and Research) that provides features for both knowledge management and decision support and enables real-time collaboration between actors in a CSE project. RADAR allows analysts to capture problems and goals, solution approaches, uncertainty, and insights. It then helps decision-makers to discover need for research as well as actionable opportunities in a semi-automated process supported by visualizations.

Figure 4.1 visualizes how the combination of design science and OOSE methodologies addresses the design goal and prediction goal by investigating their associated knowledge questions and solving the design problem. Section 4.1 investigates the context of the design problem by surveying and mapping related tools for knowledge management, project management, requirements management, and decision support. Section 4.2 conducts treatment design by creating a system design on top of CRDM and CORTEX while utilizing off-the-shelf components chosen from related tools. This yields the RADAR decision support system and addresses the artifact design goal. Section 4.3 performs treatment validation using technical action research and focuses the effects of RADAR in the context of projects applying CORTEX, addressing the prediction goal. In OOSE terms, this represents system design based on the Analysis Model and leading to the System Design Model, followed by corresponding model validation.

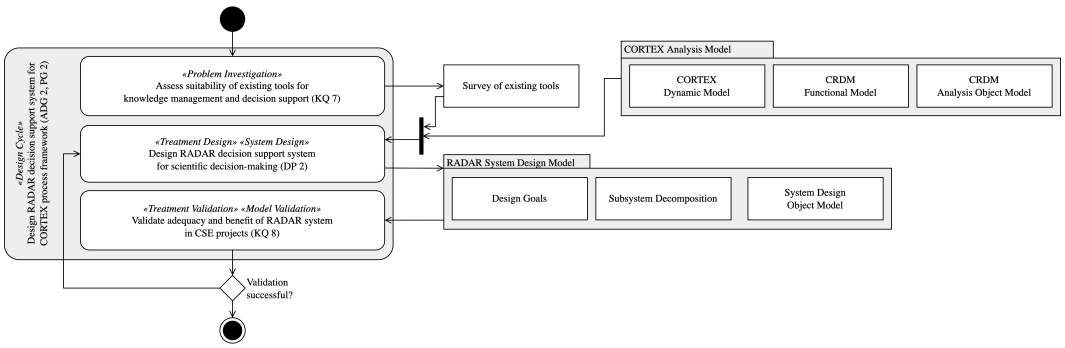


Figure 4.1: Overview of activities and entities in the third design cycle of the research project, yielding RADAR decision support system and a prediction about its effects.

4.1 Suitability of Existing Tools

The design cycle of RADAR starts with problem investigation to further understand the problem context before designing a treatment [63]. Additionally, OOSE proposes the use of off-the-shelf components as a form of reuse in order to help with the realization of a system design [36]. This is realized by investigating knowledge question 7: "Which existing tools fulfill the requirements from the Continuous R&D ecosystem and could therefore serve as the basis for RADAR?" This entails the following research questions:

1. Which types of tools are already available?
2. Which tools are widely used in each category?
3. How do these tools rank in terms of the presumed requirements?
4. In what way could they be used in treatment design?

Figure 2.2 depicts the sequence of steps in applying the Comprehensive Literature Review (CLR) framework [64] to the investigation, categorization, and evaluation of available tools regarding their suitability for knowledge management and decision support in CORTEX. Section 4.1.1 describes how the tool categories of decision support, knowledge management, project management, and requirements management were surveyed. Section 4.1.2 presents the selected representative tools for each category and section 4.1.3 rates them according to the requirements satisfaction with regards to the Continuous R&D ecosystem.

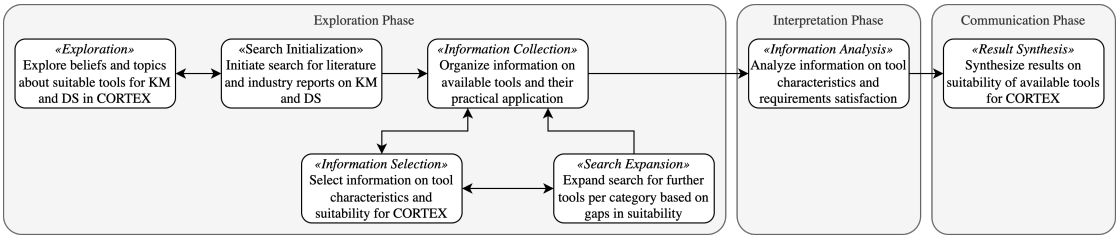


Figure 4.2: Process of research and review of tools for knowledge management (KM) and decision support (DS) the Comprehensive Literature Review (CLR) framework [64].

4.1.1 Tool Survey Methodology

Elicitation of related tools for knowledge management and decision support uses a process similar to the review of related process models for CORTEX (see section 3.1.1):

1. Based on Continuous R&D requirements, determine relevant categories of tools.
2. Search for related tools in those categories.
3. Select tools based on their practical usage and relevance for RADAR.
4. Rate tools based on their fulfillment of Continuous R&D requirements.

RADAR aims to integrate two paradigms: knowledge management and decision support. Knowledge management (KM) systems focus on storing and organizing information in order to search it efficiently and find answers to questions. Decision support (DS) systems focus on increasing efficient decision-making and improving the quality of those decisions. Integrating decision support with knowledge management supports interacting with information as well as enhances the quality of support compared to each system alone [129, 93, 130]. Potential tool categories are surveyed deductively from literature on knowledge management and decision support as well as inductively from data on practical tool usage in agile software development, specifically the annual State of Agile reports[1]. For each category, a representative tool is chosen based on criteria derived from the functional and nonfunctional requirements of Continuous R&D:

- Prevalence: Tool should be in widespread use, ideally continuously over time (i.e., not a trend).
- Flexibility: Tool should support multiple use cases with a general feature set.
- Specialization: Tool should support KM and DS with use-case-specific features.
- Customizability: Tool should be adaptle to Continuous R&D through configuration and extension.
- Usability: Tool should be intuitive to learn and easy to apply.
- Availability: Tool should be freely accessible in terms of price/license and platform.

Fulfillment of requirements by the representative tools is rated both on a theoretical and a practical basis. First, documentation and reports on each tool are reviewed and matched against requirements. Second, a short case study evaluates all of them in parallel in practical use for one month. This serves to ground the rating in practical experience specific to the context of Continuous R&D. Rating uses a five-point Likert scale from 0 ("not fulfilled") to 4 ("completely fulfilled").

[1] VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`.

4.1.2 Tools Used by CSE Projects

Analysis of widely used tools according to the State of Agile reports produced five categories of applications that agile projects use for knowledge management and decision support: Task management, spreadsheet, wiki, project management, and requirement management. Two additional categories were added based on the hypothesis that they would be useful: Mind mapping and modeling. Graph-based tools have long been used for managing and structuring knowledge for problem solving. For instance, gIBIS was developed as a graphical notation for the IBIS process model [78, 131] and implemented in tools like Compendium[2]. Visualization offers advantages when managing information for decision support: It provides a better perception of the problem and can be used for information sharing [132, 133].

Figure 4.3 visualizes the identified tool categories and classifies them based on whether they represent information visually or textually and whether they are general-purpose tools or specialized applications. Table 4.1 shows the selected categories, the respective tool chosen to represent it, and the rationale for selecting the particular tool.

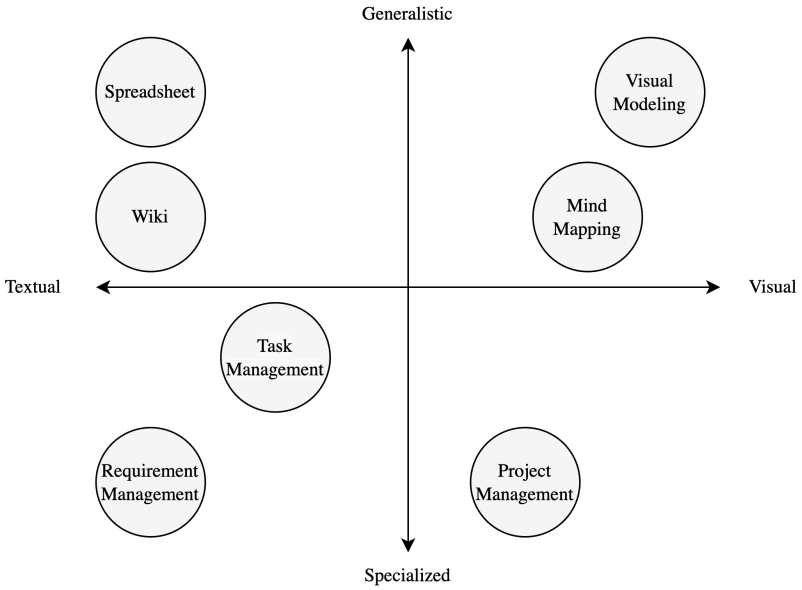


Figure 4.3: Classification of relevant tool categories according to representation of information and flexibility of feature set.

[2] http://projects.buckinghamshum.net/compendiuminstitute/

Category	Selected Tool	Rationale
Task Mgmt.	Atlassian Jira	Widespread use (74 %), customizable
Spreadsheet	Microsoft Excel	Still prevalent (74 % to 65 %), flexible
Wiki	Atlassian Confluence	Widespread use (62 %), flexible
Project Mgmt.	OmniPlan	Popular choice (43 %), specialized
Requirement Mgmt.	Siemens Polarion	Usage in projects of 46 %, specialized
Mind Mapping	FreeMind	Flexible, easy to use
Modeling	Draw.io	Flexible, freely accessible

Table 4.1: Selected tool per category and corresponding rationale for tool choice.

4.1.3 Satisfaction of Requirements

After tool evaluation, a combined score was calculated as the weighted average between functional and nonfunctional score (weighted by the respective number of maximum points). The selected tools achieved a widespread range of scores with the combined score ranging from 33 % (OmniPlan) to 96 % (Atlassian Jira). Table D.1 lists individual ratings on functional and nonfunctional requirements for each tool. Figure 4.4 shows the individual and combined scores per tool. Atlassian Jira emerges as the winner, closely followed by Siemens Polarion with a combined score around 90 %. Excel, FreeMind, Draw.io, and Atlassian Confluence rank in the middle with a combined score around 50–60 %. OmniPlan takes last place with a combined score of 33 %, which might be surprising for a tool dedicated to project support.

As the scores indicate, not all tool categories match equally well with the needs for knowledge management and decision support. Figure 4.5 visualizes the rating of each tool by size and distributes them on the same axes as fig. 4.3 (flexibility of feature set and representation of information). This reveals a correlation of tool category and score for the purposes of RADAR: Tools that are very specialized for project support and focused on detailed capture of information (either tasks or requirements) achieve the highest score. Surprisingly, the next best score is achieved by tools that represent the exact opposite, namely generalistic tools for visual representation of information. Tools outside of these two clusters score the lowest with a significant distance.

This concludes the validation with two deductions:

1. RADAR should combine the functionality of information-driven tools for knowledge management and visual tools for decision support to realize the benefit of both categories.

2. RADAR could use Atlassian Jira as its basis since it ranks the highest, not least because of its focus on task management which fits well with CRDM/CORTEX, and since it is very customizable through configuration and extension.

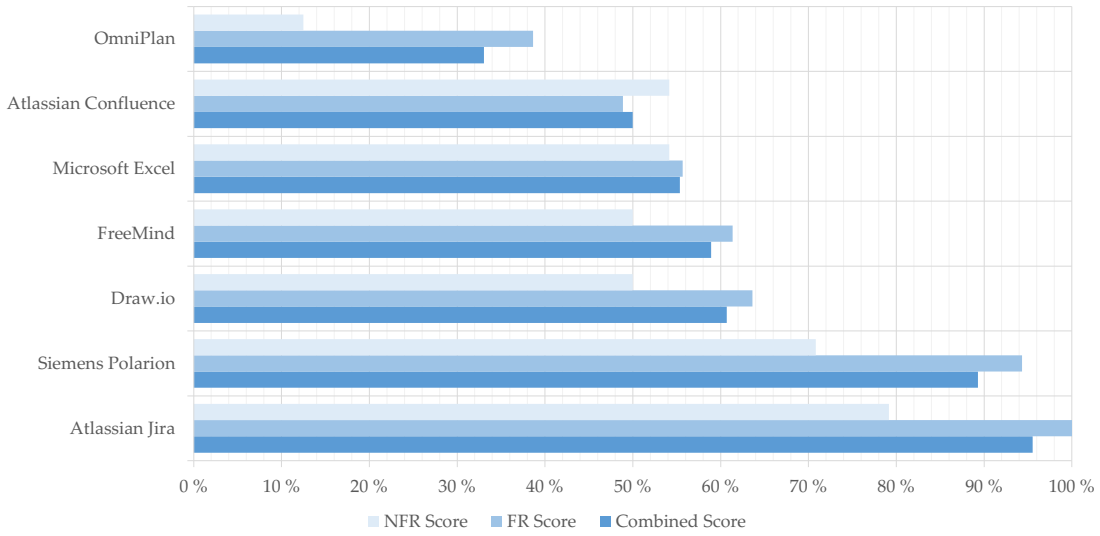


Figure 4.4: Rating of selected tools based on functional and nonfunctional requirements, sorted by combined score.

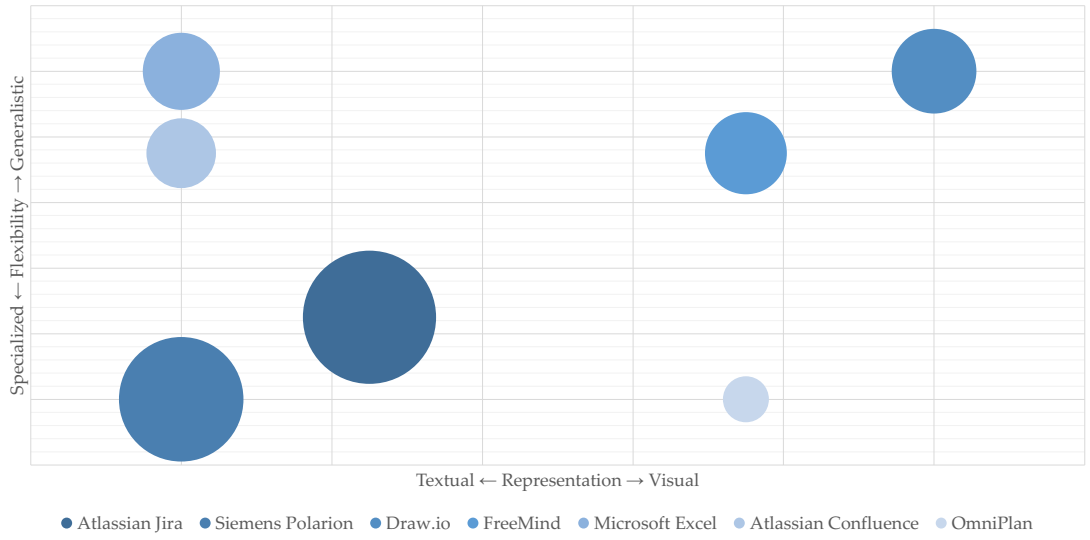


Figure 4.5: Rating of selected tools segmented by representation and flexibility, score represented by size.

4.2 Design of RADAR System

After investigating the problem context of decision support in CSE and reviewing related tools for potential reuse, the next step is the treatment design of RADAR [63]. Using CRDM and CORTEX as a basis in combination with the insights from problem investigation, this tackles design problem 2: How to design a decision support system that captures knowledge and uncertainty so that CORTEX can be applied more easily? This involves the following research questions:

1. Which design goals describe the desired qualities of the system design?
2. How to cover the elements of the CRDM and CORTEX models in a top-level system design?
3. How to decompose the top-level system design into subsystems?
4. Which related tools can be considered as off-the-shelf components for reuse in each subsystem?
5. How to design each subsystem that cannot be (entirely) implemented using off-the-shelf components?

In OOSE terms, system is based on the Analysis Model and produces a System Model through iterative subsystem decomposition and refinement [36]. Section 4.2.1 starts the process by defining design goals that govern the design of the system. Section 4.2.2 describes the system architecture that captures the elements of the CORTEX Analysis Model and decomposes them into subsystems. It also includes a corresponding selection of suitable off-the-shelf components and mapping of software to hardware components. Sections 4.2.3 to 4.2.5 describe the subsystems for knowledge management, decision support, and project management. Section 4.2.6 describes the object model required for integrating with the selected off-the-shelf components.

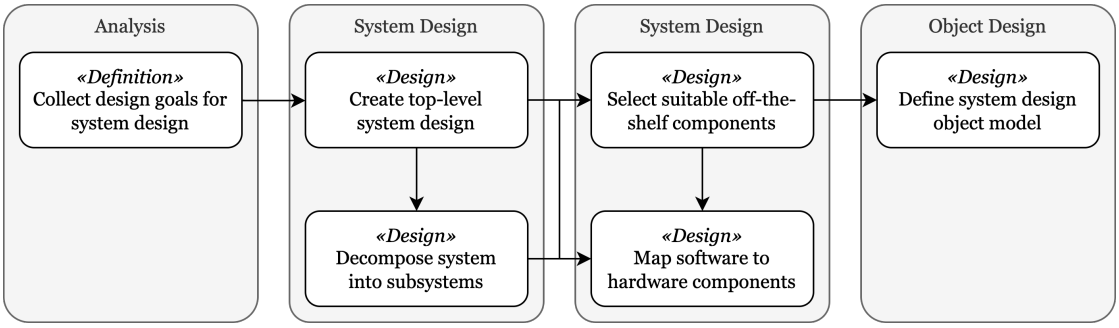


Figure 4.6: Process of designing the RADAR decision support system consisting of overall architecture, subsystems, and object model.

4.2.1 Design Goals

Bruegge and Dutoit [36] recommend to identify design goals as the basis for system design: "[They identify] the qualities that our system should focus on. Many design goals can be inferred from the nonfunctional requirements or from the application domain. Others will have to be elicited from the client. It is, however, necessary to state them explicitly such that every important design decision can be made consistently following the same set of criteria." The following design goals are based on their classification and definition of criteria. Criteria that are not critical to this research project are omitted.

Performance criteria Speed and space requirements imposed on the system based on user expectations and available resources.

1. Response time: User requests must be acknowledged and answered as quickly as possible to allow for efficient decision support.

2. Throughput: The system must be able to handle all members of a team issuing requests in parallel. We anticipate between 10 and 100 users in an organization.

Dependability criteria Minimization of system errors or crashes as well as their consequences, including tolerated frequencies of and the expected behavior in exceptional cases.

3. Robustness: The system must be able to handle any invalid user input.

4. Reliability: The system must behave exactly as specified in the requirements (section 2.1). If requirements don't specify precise behavior, it must adhere to the CRDM scenarios (section 2.2.1) and CORTEX workflows (section 3.2).

5. Availability: The system must be available all the time to sensibly support an entire project organization. Only short periods of downtime are tolerable.

Cost criteria Cost for development, deployment, and administration of the system, including managerial considerations, expectations regarding compatibility with other systems, and costs of introducing the system to their intended user base.

6. Development cost: Developing the initial system must be as cheap as possible in order to quickly evaluate its use in supporting the CORTEX process.

7. Deployment cost: Installing the system and training the users must be as cheap as possible in order to quickly evaluate its use in supporting the CORTEX process.

8. Administration cost: Projects can only invest minimal effort in administrating the system, therefore it should require very few administrative tasks.

Maintenance criteria Permissible effort required for keeping the system functional after deployment as well as changing it, including extensions of its functionality, adaptions to different application domains, and ports to different technology.

9. Extensibility: We anticipate future work in extending RADAR, therefore upgrading must be possible with minimal effort.
10. Modifiability: We anticipate future work in modifying RADAR, therefore upgrading must be possible with minimal effort.
11. Adaptability: The system must be able to adapt to different application domains with little or no effort.

End user criteria Desirable qualities from the point of view of the system's users that are not already covered by performance and dependability criteria, including tolerated difficulty to learn the system and accomplish tasks using it.

12. Utility: The system must perfectly fit CORTEX workflows and support all participating actors appropriately. Trade-offs should be made in favor of supporting the work of the user.
13. Usability: All features must be usable by users from different backgrounds intuitively with little room for error. Trade-offs should be made in favor of ease of use.

4.2.2 System Architecture

Following the OOSE framework for system design, the system architecture is initialized by designing a top-level architecture capturing the elements from the analysis model [36]. This design is subsequently decomposed into subsystems by separating responsibilities and design concerns. RADAR consists of subsystems for knowledge management, decision support, and project management. The system design addresses design goals by selecting suitable off-the-shelf (OTS) components for each subsystem where applicable. This especially aims to reduce implementation costs and promote software reuse. Software components are then assigned to suitable hardware components. RADAR continues uses Atlassian Jira as the basis for its features, extending and customizing them as necessary.

The System Design Document described by OOSE contains further sections that are not addressed here: Global control flow, boundary conditions, access control, and persistent data management. Since RADAR's design is based on Atlassian Jira, these details are implicitly determined by Jira's system design. Atlassian provides a detailed documentation[3] for the details of these aspects of the system.

4.2.2.1 Subsystem Decomposition

Subsystems are separate, substitutable parts of the system with well-defined interfaces, which encapsulates the state and behavior of contained classes [36]. Subsystems are decomposed to that functional and nonfunctional requirements are grouped in a sensible way. Also, the system structure should allow addressing design goals in each subsystem, e.g., by selecting suitable technologies and reusing existing components. RADAR consists of three subsystems: knowledge management, decision support, and project management. Their responsibilities are defined as follows:

- **Knowledge Management:** Capture entities and information, support prioritization/estimation/validation, guide use of research results (FRs 1, 2, 4 and 9).
- **Decision Support:** Support discovery of opportunities for problem solving, identify need for research or implementation, facilitate learning (FRs 3, 5, 6 and 10).
- **Project Management:** Connect research and decision support to execution, support experimentation using suitable methods (FRs 7 and 8).

Figure 4.7 visualizes the subsystem decomposition of the RADAR system. In addition to interfaces between systems it depicts which actors use which components. (For simplicity, interfaces only denote if data can be read or written; the respective data can be inferred from each component, e.g., knowledge management system allows reading and writing knowledge entities).

[3] `https://confluence.atlassian.com/jira`

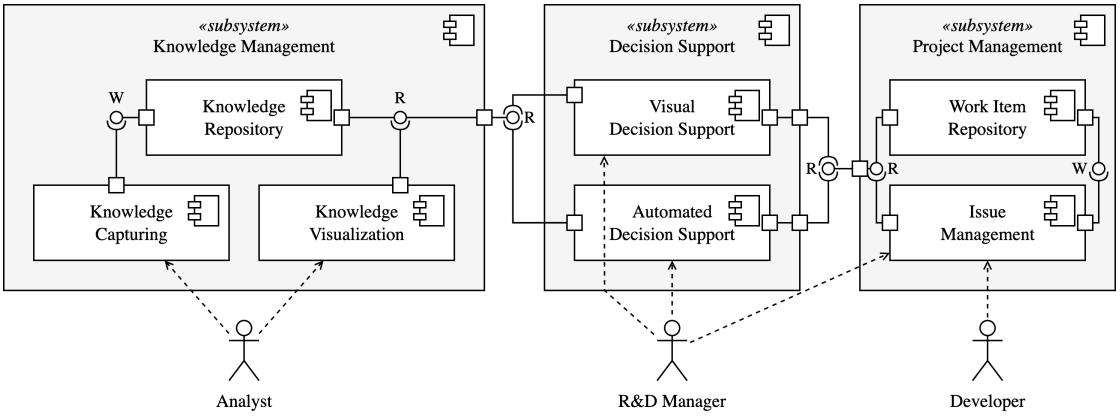


Figure 4.7: Subsystem decomposition into knowledge management, decision support, and project management; "R" and "W" denote interfaces for reading and writing data.

4.2.2.2 Use of Off-The-Shelf Components

"Make or buy" is an important decision in software engineering that makes trade-offs between factors such as initial development costs, ongoing maintenance costs, control over fulfillment of requirements, and flexibility with extension and adaption [134, 135, 136]. Harris [137] estimates that maintenance costs for developed software are about 55 % of the total cost of developing the product. RADAR aims to reduce both development and maintenance costs and therefore selects Atlassian Jira as the basis. Jira was carefully selected for implementing Continuous R&D and offers a solid basis for RADAR's features.

RADAR utilizes standard functionality of Jira and extends and customizes it as necessary. Jira offers multiple powerful mechanisms for customization and extension, based on a flexible object model, query language, data API, and plugin system. Atlassian provides a detailed documentation[4] for the details of these aspects of the system. The following list describes the most important components of Jira and the possibilities for reuse they offer. Each subsystem will define how its respective requirements are addressed by Jira and which extensions and customizations it uses in the subsequent sections.

Issue types (standard & custom) Issues are the core of Jira and capture information as well as state of the issue. Issues can carry links to other entities, both issues or external resources. Jira offers a set of standard issue types for agile development such as epics, user stories, tasks, and sub-tasks. Additionally, custom issue types can be defined allowing individual configuration of fields, workflows, and links.

Issue workflows (standard & custom) Issues are transitioned through states of a workflow, which defines which states exist for an issue type. Workflows can optionally impose restrictions on allowed transitions within the state machine. Workflows can also contain hooks to automatically perform actions on certain transitions. Jira offers default

[4] `https://confluence.atlassian.com/jira`

workflows for agile development that can be modified or replaced with completely custom workflows.

Data fields (standard & custom) Issues carry information in data fields which can have different types such as text, dates, files, or checklists. Jira offers a set of standard fields to capture information that is relevant for development and issue tracking (e.g., title, description, due date). Additionally, it supports the definition of custom fields which can use standard field types or additional field types offered by extensions. Screen schemes control which fields are displayed based on issue type, workflow state, and performed action. Again, there are standard options but they are fully customizable.

Issue links (standard & custom) Issues can be connected through links that carry semantics by using different link types. Jira offers a set of standard link types for development and issue tracking (e.g., blocked by, requires, resolves). It also allows the definition of custom link types for modeling $n : 1$, $1 : n$, or $n : m$ relationships with specific semantics. Issue type link schemes control which link types are available for each issue type.

Issue search Jira offers multiple options to browse existing issues (e.g., dashboards, backlog view, Kanban boards, lists). Exploration of the knowledge base is possible through full-text search as well as the structured Jira Query Language (JQL) for advanced queries. For example, JQL queries can query different fields across issue relationships and apply logical conditions. Searches can be stored as filters and used in dashboards, backlog views, and Kanban boards.

Issue Links Viewer extension In addition to the built-in methods for browsing issues and search results, the extension Issue Links Viewer[5] offers an even more powerful structural overview by visualizing issues and links as a graph. It allows visualizing search results as graphs and viewing issue information directly from the graph.

ScriptRunner extension In addition to the built-in methods for querying and updating the issue database, the extension ScriptRunner[6] expands Jira's feature set with powerful scripting abilities. Scripts can be used for multiple use-cases that are relevant to RADAR: automatically creating, updating, or transitioning issues; performing more complex searches than standard JQL; algorithmically extracting data from the issue graph and external sources; providing scripted data fields with auto-calculated values.

4.2.2.3 Hardware/Software Mapping

The hardware/software mapping describes how subsystems are assigned to hardware components, which may also include off-the-shelf components [36]. The base system

[5] `https://marketplace.atlassian.com/apps/1216207/issue-links-viewer`
[6] `https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira`

Jira is a web application with a client-server architecture. In this architecture, a central server serves requests from multiple clients over a defined protocol. Additionally, there is a database server hosting the Jira databases. In reality, there might be more than one application and database server or even a cluster of servers each, with the concrete configuration depending on how Jira is hosted. Figure 4.8 shows how the components of the RADAR subsystems are deployed across the client and server execution environments provided by Jira. As mentioned initially, Jira's system design takes care of central concerns like authentication and global control. Subsequent sections will show how the components of each subsystem are connected internally.

The user interface components are implemented as part of the Jira web application executed in the user's web browser. Each user interface component communicates with the respective service within the Jira application. This connection uses HTTPS ("HyperText Transfer Protocol Secure") as is standard for modern web applications. On the application level, Jira provides a REST API ("Representational State Transfer") for accessing data and functionality[7]. Depending on the subsystem and component, requests can read or write data to and from the service. Extensions can interface with Jira on the client and server using the plugin system[8] and also expose their own data and functionality via REST APIs over HTTPS. Data of repository components is stored in the database using the database schema defined by Jira or the customizations. Services can connect to the database via SQL ("Structured Query Language") to read and write data.

[7] `https://developer.atlassian.com/server/jira/platform/rest-apis/`
[8] `https://developer.atlassian.com/server/framework/atlassian-sdk`

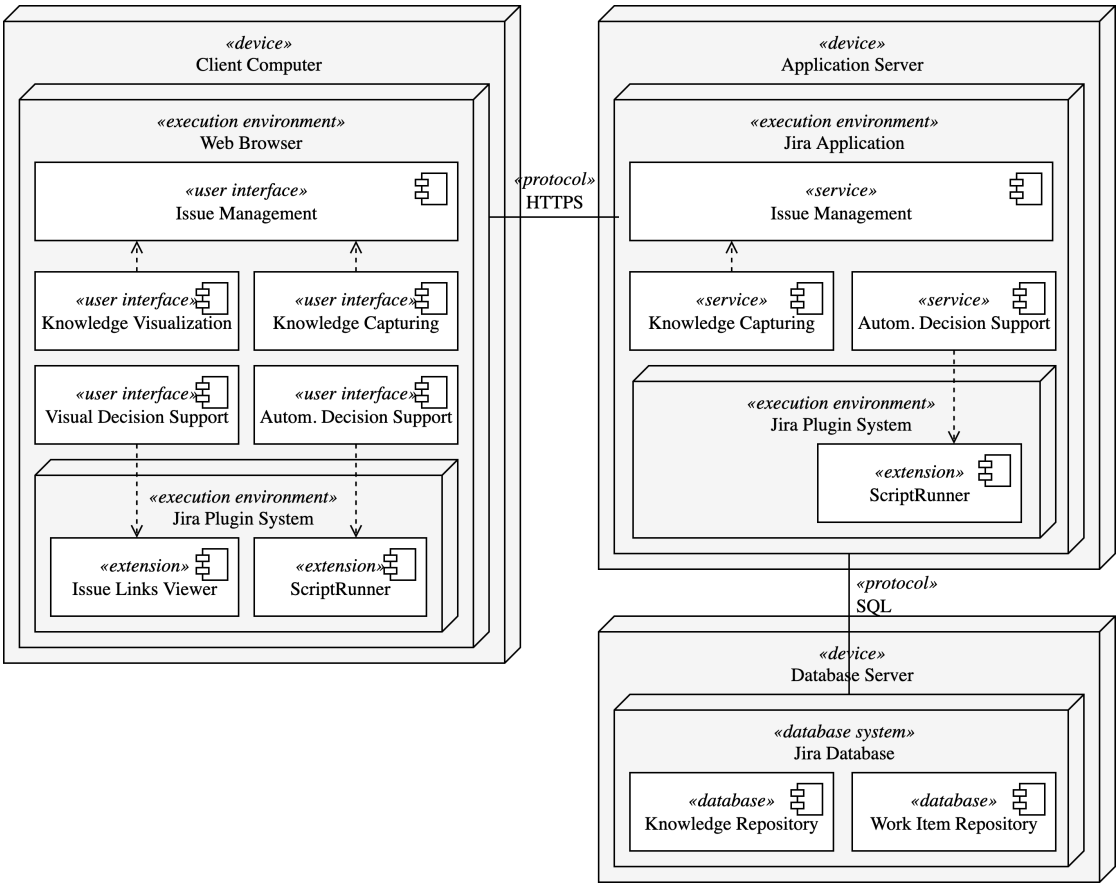


Figure 4.8: Hardware-software mapping assigning software components to execution environments in a client-server architecture.

4.2.3 Knowledge Management Subsystem

Knowledge management forms the basis for decision support in Continuous R&D. The respective subsystem is responsible for capturing entities as well as their connections and attributes as the problem and solution domain are explored. Additionally, it supports the validation of problems, solutions, and connections by allowing users to inspect the currently known information and the inherent uncertainty. In doing so, it not only manages the knowledge built up by the project over time but also supports analyzing it to extract further insights and guides its use for further activities [138]. Table 4.2 lists how the additional functional requirements of Continuous R&D that are relevant for the knowledge management subsystem are implemented using standard Jira functionality as well as customizations and extensions.

Functional requirement	**Jira functionality**
FR 1: Capture problems and solutions	Custom issue types, custom link types
FR 2: Estimate benefit, cost, uncertainty of entities	Custom fields
FR 4: Validate known problems, solutions, and connections	Issue search and filters
FR 9: Guide use of research results in decision-making	Issue Links Viewer extension, custom screen schemes

Table 4.2: Fulfillment of functional requirements for knowledge management to suitable standard functionality or extension mechanisms in Jira.

Knowledge Repository stores all forms of knowledge, i.e., all types of facts, their interconnections, and associated uncertainty factors. A repository is a separate component for storing and accessing data, which decouples persistence from other concerns and allows multiple other components to access this data in a central place and controlled fashion [36]. Since it uses Jira as its base system, it also stores a detailed change history of entities and support fine-grained access control. *Knowledge Capturing* continuously updates the knowledge repository with incoming information about entities, uncertainty, priorities, and estimates. *Knowledge Visualization* provides access to collected knowledge and supports investigating it through visual exploration as well as targeted search. *Knowledge Capturing User Interface* and *Knowledge Visualization User Interface* access the information provided by the respective application services and allow interacting with knowledge management features. Figure 4.9 visualizes the components of the subsystem and their connections.

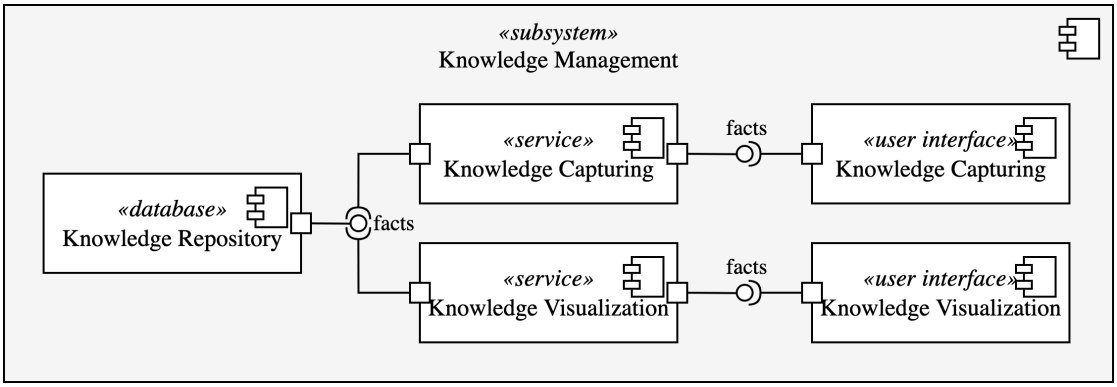


Figure 4.9: Subsystem for knowledge management with separate application service and user interface components.

4.2.4 Decision Support Subsystem

Decision support draws on the knowledge generate in research and provides aids and insights for the problem solving process. RADAR defines both a visual and an automated version of decision support. The former uses data visualization to present the available information in a way that helps decision-makers recognize important aspects in the knowledge base. The latter uses data mining to analyze available information and extract insights on its own, presenting decision-makers with proposals for potential decisions. Its goal is not to replace the human decision-maker but to augment them by pre- or post-processing information and thus making large amounts of data more manageable. This yields a semi-automatic process, utilizing both human and artificial intelligence where they perform best – an approach called "adaptive autonomy".

Table 4.3 lists how the additional functional requirements of Continuous R&D that are relevant for the decision support subsystem are implemented using standard Jira functionality as well as customizations and extensions.

Requirement	**Jira functionality**
FR 3: Discover and seize opportunities for problem-solving.	Issue search and filters, ScriptRunner extension
FR 5: Identify need for further research.	Issue Links Viewer extension, ScriptRunner extension
FR 6: Identify potential for implementation.	Issue Links Viewer extension, ScriptRunner extension
FR 10: Facilitate learning from research.	Issue search and filters, custom screen schemes

Table 4.3: Fulfillment of functional requirements for decision support to suitable standard functionality or extension mechanisms in Jira.

Visual Decision Support and *Automated Decision Support* fetch the existing knowledge stored in *Knowledge Repository* to support decision-making. *Visual Decision Support* allows exploring the knowledge base by visualizing hierarchies and networks of entities and rendering additional information stored in data fields. *Automated Decision Support* aims to perform the CORTEX problem solving process, traversing the entities in the knowledge base and detecting need for research as well as opportunities for implementation. Decisions from both components may generate work items in *Work Item Repository*. *Visual Decision Support User Interface* and *Automated Decision Support User Interface* access the information provided by the respective application services and allow interacting with decision support features. Figure 4.10 visualizes the components of the subsystem and their connections.

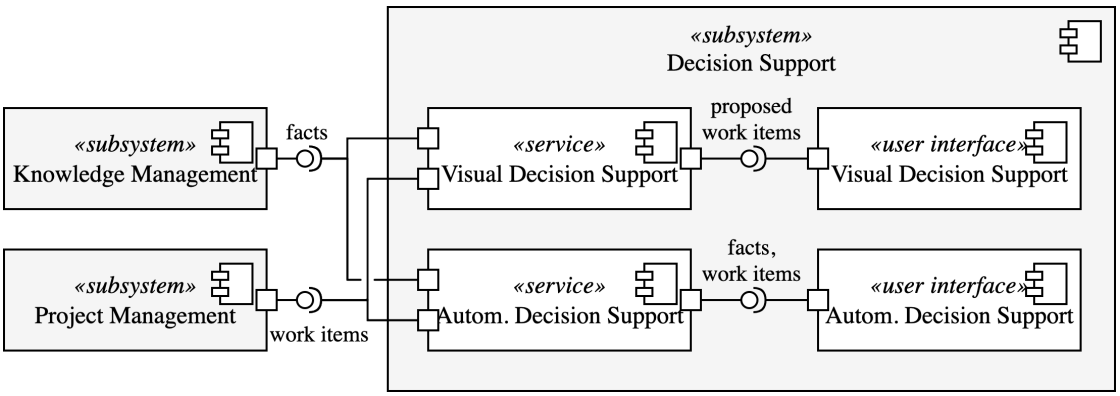


Figure 4.10: Subsystem for decision support with separate application service and user interface components.

4.2.5 Project Management Subsystem

All decisions generate work items, either research tasks or implementation tasks. The project management subsystem is responsible for storing, organizing, and tracking these work items. Based on Jira, it provides all functionality necessary for issue management, extending into project management with planning and controlling features. Table 4.4 lists how the additional functional requirements of Continuous R&D that are relevant for the project management subsystem are implemented using standard Jira functionality as well as customizations and extensions.

Requirement	**Jira functionality**
FR 7: Support experimentation using suitable research methods.	Custom issue types
FR 8: Support experiments in laboratory and real world.	Custom workflows, custom screen schemes

Table 4.4: Fulfillment of functional requirements for project management to suitable standard functionality or extension mechanisms in Jira.

Work Item Repository stores work items , i.e., all information from issues, data fields, issue links, etc. *Issue Management* component accesses this information and executes application logic required for managing the work items (create, update, delete) as well as transitioning them through their respective workflow, optionally execution automations triggered by workflow hooks. *Issue Management User Interface* accesses the information provided by the application service and allows interactions with project management features. Figure 4.11 visualizes the components of the subsystem and their connections.

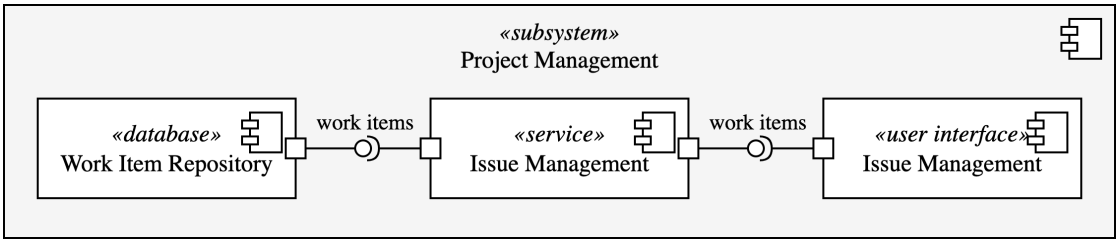


Figure 4.11: Subsystem for project management with separate application service and user interface components.

4.2.6 System Design Object Model

The system design object model refines the analysis object model with details from the subsystem decomposition and describes according interfaces [36]. RADAR's object model describes how the CRDM object model is integrated with Jira's object model to support a seamless implementation of the RADAR system design.

Jira's object model centers around a *Project* which contains its *Issues*. All information is attached to issues which serve as a highly configurable information management system. The flexibility of the object model is based on the decoupling of information and configuration. Information classes capture projects, issues, links, data fields, issue operations, screens, transitions, and workflow status. Configuration classes associate issue types with field configurations, screen schemes, workflows, and link types; they also associate issue operations and workflow transitions with screens. Figure 4.12 shows the Jira object model with configuration classes interlaced between the project and other information classes. RADAR extends Jira with custom types to cover the object model and feature set of Continuous R&D. Figure 4.13 shows how the default classes are extended and how the customizations correspond to each other.

- *Entities* (problem, solution, connection) are implemented as custom issue types.
- *Attributes* (benefit, cost, suitability) are implemented as custom fields.
- *Hierarchy* of entities is supported with custom link types for $1 : n$ relationships (problem contains problems, solution contains solutions).
- *Connections* are $n : m$ relationships and additionally require a custom link type (connection connects problem, connection connects solution).
- *Uncertainty* is implemented as a custom issue type so its type and degree can be captures as custom data fields and it can be connected to entities ("fact contains uncertainty").
- *Research tasks* are implemented as custom issue types with a custom link type (research task reduces uncertainty).
- *Implementation tasks* are similarly implemented as custom issue types with a custom link type (implementation task implement solution).

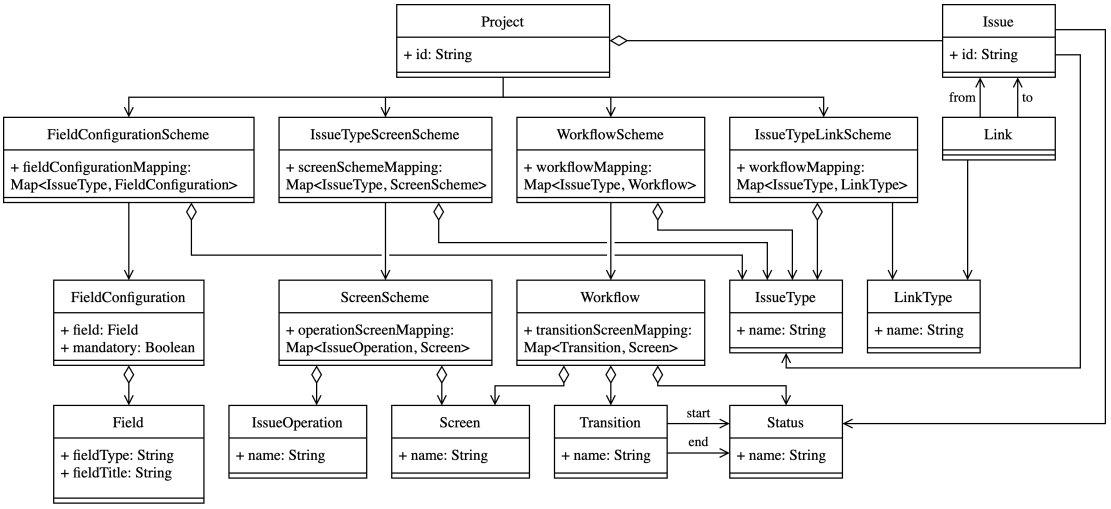


Figure 4.12: Object model of built-in classes and associated configuration and customization mechanisms of Jira.

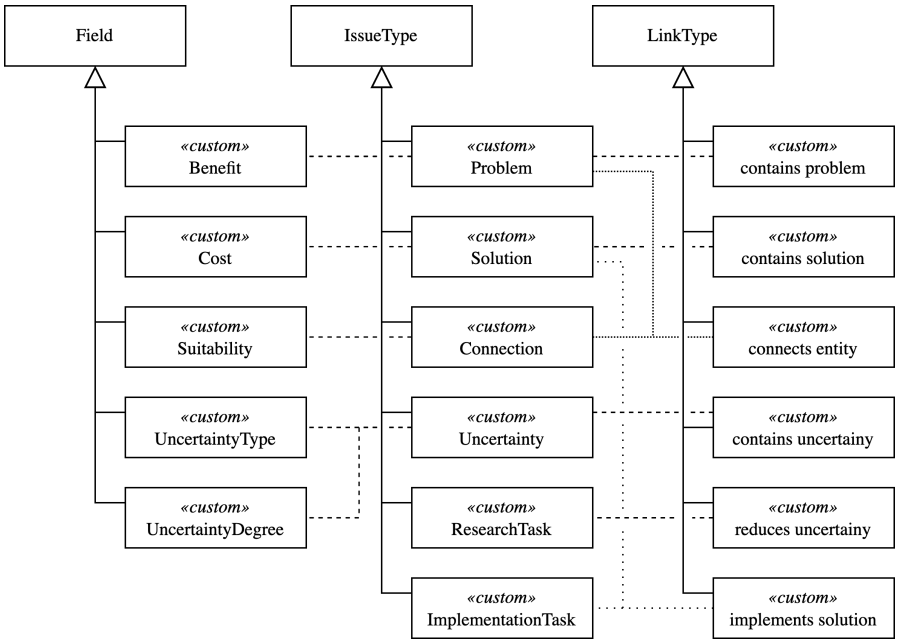


Figure 4.13: Extension of Jira object model with custom issue types, link types, and fields to adapt it to CORTEX.

4.3 Validation of RADAR Effects

After treatment design, the RADAR design cycle concludes with treatment validation to predict the effects of the artifact in the problem context. RADAR is validated regarding its effects in the real-world problem context by investigating knowledge question 8: *Does RADAR help with applying each aspect of CORTEX?* It therefore also concludes prediction goal 2: *Predict the benefits of RADAR when applying CORTEX.* This implies a combination of requirements satisfaction questions regarding the behavior and performance of the treatment as well as effect questions regarding the interactions and effects of the treatment [63]. Applied to RADAR, this entails the following research questions:

1. Does RADAR support the knowledge management activities in CORTEX?
2. Which events and activities are captured in RADAR over the course of a project?
3. Does RADAR support the decision-making activities in CORTEX?
4. What is the distribution of implementation and research decisions in RADAR?

In terms of OOSE, this represents validation of the RADAR System Model. In contrast to the separate validation activities for CORTEX, one case study covers both its correctness with respect to its requirements (verification) and its actual benefits for the application of CORTEX (validation). Figure 4.14 visualizes how the case study uses technical action research with a prototypical implementation of RADAR as its analogical representation. Section 4.3.1 describes the prototyping and technical action research methodology of the case study. Section 4.3.2 describes benefits of RADAR for knowledge management activities in CORTEX and illustrates the temporal progress of these activities. Section 4.3.3 describes benefits of RADAR for decision-making activities in CORTEX and illustrates the interplay between implementation and research. Appendix D.2 provides more details on the execution of the case study, both regarding implementation and evaluation.

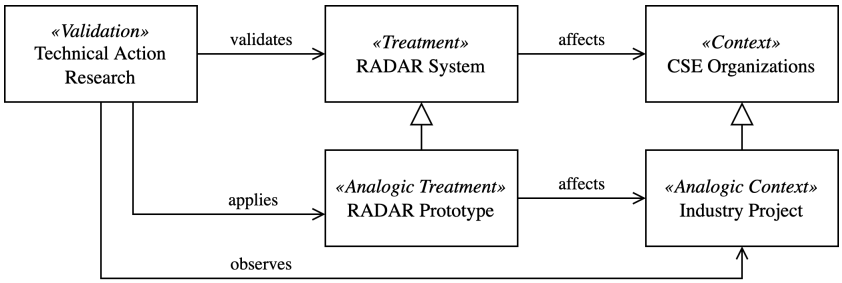


Figure 4.14: Validation of RADAR using a prototype in an industry project.

4.3.1 Case Study Methodology

PANDA is field-tested using an observational case study in a business environment which is then evaluated using an end user survey [139]. The goal is to assess how the decision support system performs in a real-world context. The case study observes whether the purposed-designed features of the system benefit knowledge management as well as decision support. Data is collected from two main sources: Observations on RADAR usage collected during the project case study, and data collected in RADAR over the course of the studied project. Findings should validate the suitability of the system design for the context of Continuous R&D and potentially reveal potential weaknesses or gaps. The case study is observational because RADAR has already been designed to fulfill the functional and nonfunctional requirements of Continuous R&D, implement the CRDM model, and support the CORTEX process model. Validation therefore uses these aspects as criteria for determining whether RADAR's design fits all of these specifications and expectations.

The case study is executed as technical action research (TAR) [105] in the context of an industry project within a large organization, specifically an automotive original equipment manufacturer (OEM) in collaboration with an IT consulting company. The subject of the case study is a project with 60 participants divided into four teams and distributed across three countries, working independently and often remotely. Each team consists of a team lead as well as consultants for business and IT, i.e., R&D manager, analysts, and developers. The scope of the project is software for administration and optimization of the order and manufacturing process of pre-series vehicles, a context that is extensive and complex as well as governed by success criteria on quality and cost. This project offers several challenges that suit CORTEX and RADAR, such as a complex problem domain, multiple solution alternatives, the need for asynchronous workflows, and tight constraints.

RADAR is instantiated in a *reference implementation* using a Jira instance that is self-hosted in the project environment and therefore offers full control over its setup as well as fast adaption if necessary. Figure 4.15 shows the *visual decision support component* with an issue graph that starts from a root problem, subdivides the problem, attaches solutions, shows need for research, and displays associated research and implementation tasks. Additionally, the *automated decision support component* is implemented as a semi-autonomous script. The case study collects data on the frequency of events and activities as well as the ratio of implementation to research tasks. Appendix D.2 provides further implementation details and visual examples of the reference implementation.

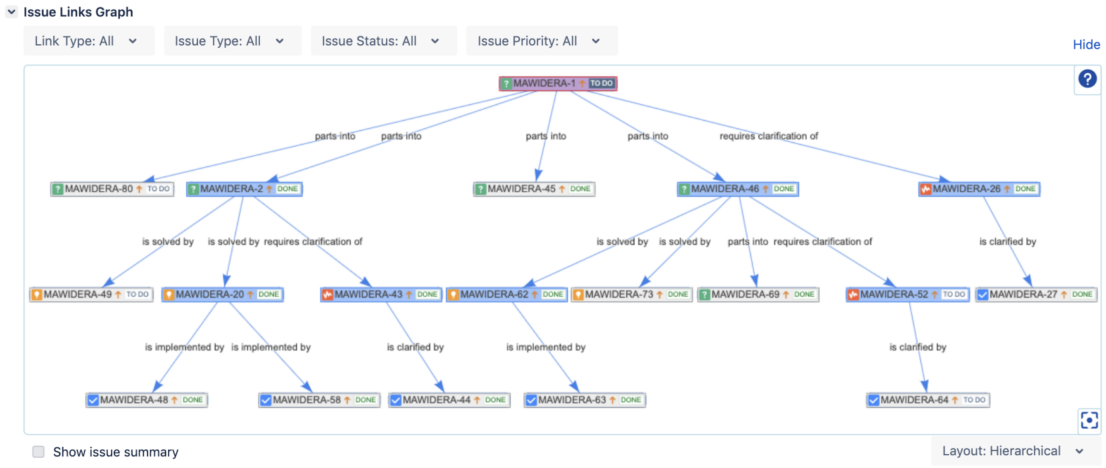


Figure 4.15: Visual decision support of RADAR showing problems, solutions, need for research, research tasks, and implementation tasks.

4.3.2 Effects on Knowledge Management

As a baseline reference, the project initially tried to apply CORTEX using pen and paper to model their problem and solution domain. However, the higher the complexity of problem domain and volatility of requirements, the more challenging it becomes to capturing them. As the amount of entities and degree of details in the knowledge base increase over time, manual knowledge management grows increasingly difficult, regardless of how much time is invested [140, 141, 142]. This confirmed the need for a purposefully designed knowledge management system.

The project made use of hierarchical entity structures by starting with "root problems" that identified interrelated contexts within the problem domain. Teams were able to focus on one problem context each, creating more focus and allowing them to build up context-specific knowledge about problems and solutions. This is in line with the principle of *Domain-Driven Design* (DDD) which puts the domain model at the forefront of software engineering and emphasizes decomposition of the domain into sub-domain and domain contexts as part of the so-called "strategic design" [117, 7]. Figure 4.16 shows the capture of entities over the course of the project with the majority of input occurring at the beginning, followed by further exploration while work items are slowly being created – a similar progression to the one seen in the CORTEX validation (sections 3.3 and 3.4).

RADAR was able to fulfill all functional requirements for knowledge management (FRs 1, 2, 4 and 9). Regarding nonfunctional requirements, the case study validated usability (NFR 1), timely reaction to new available input (NFR 6), concurrent use between and within teams (NFR 7), and simple operations (NFR 13).

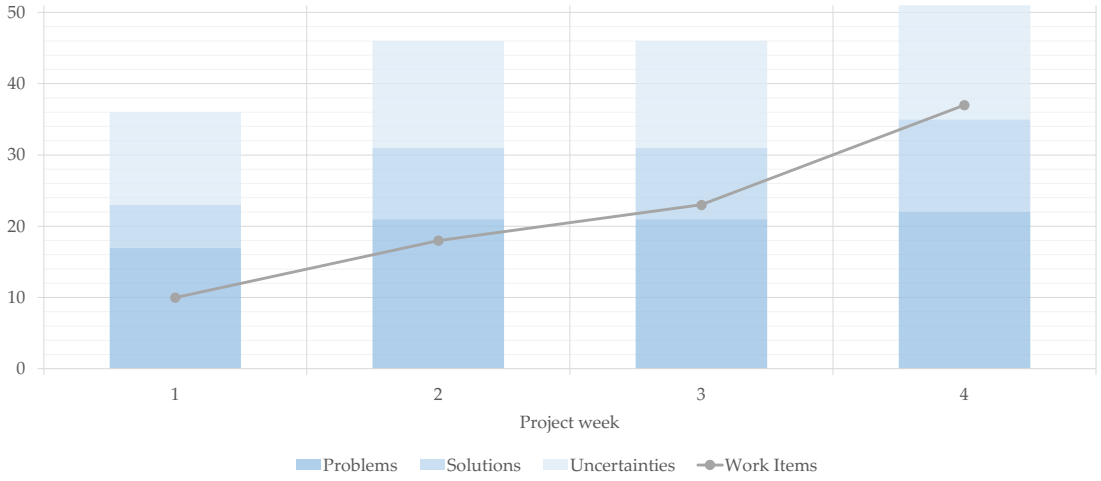


Figure 4.16: Creation of entities and uncertainties over the course of the project, overlayed with generated work items.

4.3.3 Effects on Decision Support

Even the baseline reference of "pen-and-paper knowledge management" demonstrated the value of visual decision support. Visualizing the problem and solution space provided the project with a better perception of their challenges and progress and enabled communication and collaboration [132, 143]. As the project added entities and connections, identified uncertainty factors, and started working on research and implementation in parallel, a decision support system became critical for processing the increasing amount of data. Since the knowledge base is an interconnected graph, a corresponding visual graph-based approach was the right format of conveying information and supporting decision-making [144]. RADAR's visual decision support significantly enhanced this way of using the knowledge stored in Jira, since by default Jira only allows viewing individual issues and lists of incoming/outgoing links. The automated decision support was able to suggest need for research (high-level uncertainties connected to a high value problem) as well as opportunities for implementation (solutions with a high benefit/cost ratio). It also used scripted fields to automate calculation of values across linked entities and would even be able to automatically create research and implementation tasks in a future extension.

As intended by CORTEX, the project used data for prioritization between and within problem contexts: The project discovered a total of nine "root problems" that were expanded into problem contexts. These all had different benefit/cost characteristics and had to be prioritized accordingly. For many of the problems, the project identified multiple alternative solutions and purposely issued research tasks to decide among them. The resulting data was used to opportunistically choose the solutions with the best return on invest (ROI) within each problem context. Teams also didn't have to balance effort across all potential problems in the knowledge base but only between problem areas.

They worked through them using a *divide and conquer* approach, starting from the root problem and breaking down problems into manageable entities to be investigated and solved. Figure 4.17 shows the creation of research and implementation tasks over the course of the project with research in the lead while uncertainty is being reduced but then caught up by implementation – also similar to the CORTEX validation (sections 3.3 and 3.4.

RADAR was able to fulfill all functional requirements for decision support (FRs 3, 5, 6 and 10). Regarding nonfunctional requirements, the case study validated collaboration on decisions between and within teams (NFR 2), repeatability of use in decision-making (NFR 4), resilience in ambiguous situations (NFR 5), and satisficing strategy for solutions (NFR 8).

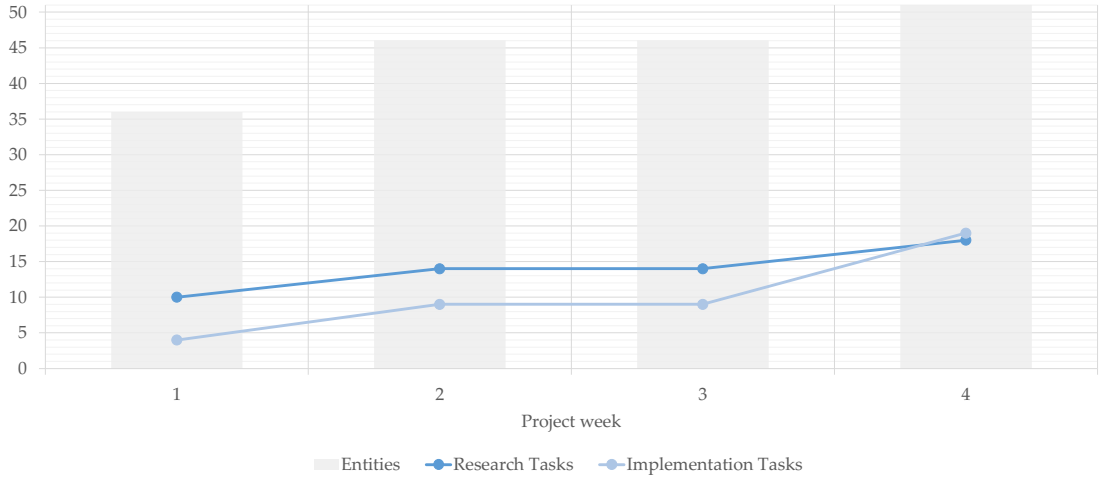


Figure 4.17: Creation of research and implementation tasks over the course of the project, imposed over generated entities.

5 Conclusion

This dissertation investigated scientific problem solving and relevant decision-making processes within the context of continuous software engineering (CSE). The motivation was to strengthen the link between business and development processes and ultimately improve the effectiveness of decisions made throughout CSE. Objectives were constructing a model of problem solving in CSE, defining a process framework and predict its effects, and designing a decision support system and predict its benefits.

These objectives defined relevant design problems and knowledge questions. According to their dependencies, a combination of design science and object-oriented software engineering (OOSE) were applied. In combination, performing requirements engineering, process modeling, and system architecture across three design cycles. This yields a model of Continuous R&D, a process framework, and a decision support system. Section 5.1 summarizes these contributions.

Results were validated using suitable methods explained in introduction. Section 5.2 discusses threats to validity of these results. All artifacts could benefit from evaluation in the field to strengthen the confidence in the results for the prediction goals obtained here. Additionally, all artifacts could be expanded with further details and extended with new concepts. Section 5.3 outlines possibilities on both areas.

5.1 Summary

Research goals and design problems were tackled by following the design science process in three design cycles, each consisting of problem investigation, treatment design, and treatment validation. Requirements engineering and system architecture activities span the three design cycles. Treatment implementation and implementation evaluation were out of scope for this research project.

Continuous R&D Model (CRDM) is a model for continuous research & development, i.e., scientific problem solving in the context of continuous software engineering. The first design cycle investigated its problem context, yielding a formalized problem statement. Requirements elicitation produced a requirements specification, subsequent analysis derived an initial analysis model. The resulting system model consists of functional model, analysis object model, and dynamic model. The second design cycle designed CORTEX, a process framework for continuous research & development based on CRDM. A gap analysis of related work in process models provided the starting point for process design. CORTEX describes an overarching problem solving process as well as workflows for problem and solution space exploration, research, and decision-making. This represents the finalized dynamic model. The third design cycle designed RADAR, a decision support system to support the CORTEX process. Existing decision support systems served as the basis for system design. This concludes system design according to OOSE by compiling a system design document.

Continuous R&D Model (CRDM) In the context of CSE, CRDM provides a coherent model for continuous research & development (knowledge goal 1), similar to how CSEPM [74] provides a model for continuous development. To investigate the problem context, the ecosystem of decision-making within continuous software engineering (knowledge question 1) was analyzed. This dissertation coins the term "Continuous R&D" for this CSE practice. The problem statement described the target environment of CSE projects, its central principles as well as the impediments of project constraints and cognitive biases. It also defined functional requirements posed by the decision-driving activities within CSE: Continuous innovation, continuous planning, continuous experimentation, and continuous improvement. The corresponding requirements specification formalized this information into a functional model by identifying actors and use cases as well as their relationships. It additionally described nonfunctional requirements, i.e., quality criteria for an approach to decision-making in CSE.

CRDM provides a dynamic model and object model of Continuous R&D (knowledge question 2). It describes the central activities of problem space analysis and solution space analysis as well as entities and relationships in three dimensions: The problem solving model covers the concepts within and between problem and solution space as well as continuous implementation. The decision support model describes decision-making, related concepts from knowledge management, particularly variability modeling. It especially differentiates the central aspects of risk and uncertainty in different forms (incompleteness, complexity, volatility, variability, ambiguity). The research model

describes exploration, investigation, and experimentation – all of which are treated as activities that can be executed independently, continuously, and asynchronously.

A case study validated understandability and applicability of CRDM (knowledge question 3) in multiple university projects. In terms of Wieringa [63], projects and hypothesis-driven development served as analogical model of the context and treatment, respectively. The model was able to cover all aspects of the project execution. Qualitative results from survey among project members showed that they were able to understand and apply the concepts and appreciate the added guidance on research and decision-making, however, with concerns about side-effects on execution efficiency. Implications of applying the model are structuring the decision-making process and acting as rationale management by making decision explicit and recording them.

CORTEX Process Framework CORTEX is a process framework for scientific problem solving in CSE (artifact design goal 1) based on CRDM. Related work was investigated to determine which existing process models fulfill the requirements of the Continuous R&D ecosystem and could therefore serve as references (knowledge question 4). Overall, eight process models were identified as a representative sample of the current state of the art. Gap analysis revealed a disconnect between processes for innovation and experimentation. This is consistent with Fitzgerald and Stol [1] who call for "BizDev", meaning closer integration between these two areas of activity analogous to DevOps.

CORTEX integrates scientific problem solving with CSE to enable more effective decision-making (design problem 1). It describes an overarching problem solving process that continuously explores problem and solution space. Over time it builds a knowledge base of facts and rates their uncertainty, then determines opportunities for implementation or need for further research. This process is based on events which, allowing it to asynchronously trigger workflows for problem and solution space exploration, research activities, and decision-making. This loose coupling suits the "chaordic" nature of CSE and allows projects to tailor the specific mechanisms (e.g., research, estimation, experimentation, prioritization) to their needs.

Validation of CORTEX aimed at predicting the impact of using it in CSE projects (prediction goal 1). A simulation experiment investigated the behavioral integrity of CORTEX and confirmed that it fulfills the requirements from the Continuous R&D ecosystem (knowledge question 5). The simulation proved the general problem solving performance of CORTEX and provided insights into its sensitivity to parameters such as the risk affinity of decision-makers. A case study validated the positive impact of CORTEX on decision effectiveness (knowledge question 6); additional anecdotal evidence strengthened this result. Additionally, it alleviated concerns about side effects raised in the validation of CRDM: Results did not show any negative impact on execution efficiency or product quality. On the contrary, CORTEX was perceived as enabling explicit knowledge management and decision-making and thereby controlling both timing and effort of research and implementation. This represents efficient resource allocation under project constraints.

RADAR Decision Support System RADAR is a decision support system for CORTEX and therefore also based on CRDM (artifact design goal 2). Existing tools and their ability to support the CORTEX process was investigated (knowledge question 7). A range of seven tool categories was identified from literature and representative tools were chosen based on industry survey data. These tools were scored on their fulfillment of the requirements from the Continuous R&D ecosystem. Task management, represented by Atlassian Jira, performed best in this comparison and was chosen as a building block for subsequent system design.

RADAR captures and visualizes both knowledge and uncertainty so that CORTEX can be applied more efficiently (design problem 2). Its system architecture describes a distributed system composed of knowledge management, decision support and project management. Jira is used as an off-the-shelf component, resulting in a client-server architecture. RADAR's system design describes data, functionality, and control flow for each subsystem. It addresses design goals by describing access control, boundary conditions, and data persistence.

Validation of RADAR focused on its benefits for applying CORTEX (prediction goal 2). A case study investigated implications for knowledge management and decision-making (knowledge question 8) using a reference implementation of the system. RADAR aided the introduction of CRDM and CORTEX by giving projects tangible artifacts and visualizing workflows. It aided with knowledge management by capturing the entire range of events, activities, and entities involved in CORTEX. It aided with decision-making by providing a structured knowledge base and allowing data visualizations such as graphs of linked issues. RADAR additionally aided with rationale management by tracking decisions over the project lifecycle and with project controlling by allowing analyses such as the ratio of implementation to research activities.

5.2 Threats to Validity

Shull et al. [122] describe that a research project needs to the describe at the very least threats to validity in the following areas in order to avoid drawing the wrong conclusions from results.

Construct validity focuses on whether the theoretical constructs are interpreted and measured correctly. It refers to the degree to which the operationalization of the measures in a study actually represents the constructs in the real world. Problems with construct validity occur when the measured variables don't correspond to the intended meanings of the theoretical terms.

Internal validity focuses on the study design, and particularly whether the results really do follow from the data. It refers to the extent to which the treatment or independent variable(s) were actually responsible for the effects seen to the dependent variable. Unknown factors may have had an influence on the results and therefore put limitations on the internal validity of the study. Typical mistakes include the failure to handle confounding variables properly, and misuse of statistical analysis.

External validity focuses on whether claims for the generality of the results are justified. It refers to the degree to which the findings of the study can be generalized to other participant populations or settings. Often, this depends on the nature of the sampling used in a study. External validity can often be a problem for controlled experiments in artificial environments where the same conditions may not hold in the real world.

Reliability focuses on whether the study yields the same results if other researchers replicate it. Problems occur if the researcher introduces bias, perhaps because the tool being evaluated is one that the researcher herself has a stake in.

5.2.1 CRDM Validation

Focus of CRDM validation was the understandability and applicability of CRDM under realistic conditions. It was validated in a case study with university projects as the analogical context and hypothesis-driven development as the analogical process. The model was applied by six project teams (four teams with 6-8 members each and two teams with 5-6 members each) over a period of three weeks, embedded in a twelve-week long capstone course program focused on modern software product development. Data was collected in a survey with both numerical and open-ended questions.

Construct Validity Validation only tested a single variant of CRDM in a single study design which could imply a mono-operation and mono-method bias. The concepts captured by CRDM as well as the analogical models used in the case study are based on fundamentals from CSE literature, suggesting that this study design is sufficient. Subjects could have guessed the purpose of the research and altered their behavior. Therefore, the case study was not announced and hidden in the normal course program: Model and process introduction as part of the regular teaching, survey as part of the regular course evaluation. In addition, subjects could have felt pressure while being evaluated, causing them to give favorable answers. Therefore, the survey was conducted after students had received their grade and were more likely to give honest feedback.

Internal Validity Since the case study was restricted to one independent variable with a within-subject design and linear execution, there is no ambiguity about the temporal precedence. For the same reason, there is no threat of diffusion, i.e., treatment effects spreading to the control group. A potential confounding variable is ability or willingness to learn. The university course is a teaching environment and therefore conducive to understanding. However, constant learning is part of any CSE environment due to the principle of continuous improvement. Characteristics of participants were not controlled for and self-selection into course and projects may have introduced selection bias. However, the course allocates students to projects to achieve a uniform distribution of characteristics like experience, skill, etc. Participants could have gained additional experience over time (maturation), skewing the results on understandability. The short periods of training (three weeks) and application (nine weeks) reduce this risk. The study might have suffered from survivorship bias with participants unable to grasp the concepts dropping out prematurely. This would have been noticed as students were registered in the course and no such events were recorded.

External Validity Generalizability could have been impacted by a study design not representative of typical CSE projects. However, the university capstone course uses real industry problem statements, teaches CSE practices, and mirrors the roles defined in CRDM. Subjects may not have been representative of the target population, introducing selection bias [145]. Even though there was no convenience sampling of projects (projects were selected randomly from a pool of eleven projects), the number of six projects was low. However, the course itself is a representative sample of innovation projects by covering,

at the time of writing, 200 projects with 2,000 participants in 20 iterations over ten years. The similarity of projects and participants, as confirmed by course management, suggests the representativeness of this sub-sample.

Reliability With only one researcher interpreting the results, the study may have suffered from confirmation bias, leading to information being interpreted in such a way that it hinders the possibility to rightfully reject a hypothesis [146]. Case study execution and interpretation of results were peer reviewed at the chair. Additionally, questions using a numerical scale were designed to leave little room for interpretation When interpreting answers to free text questions, the line of reasoning was explained. As a characteristic of technical action research, the researcher could have also influenced participants to behave according to expectations. Therefore, an intermediary interacted with the projects.

5.2.2 CORTEX Validation

CORTEX validation focused on integrity as well as effects of the process framework. A simulation experiment investigated the behavioral integrity of the problem solving process. Data was collected by taking measurements from the simulation model throughout execution. A case study in an industry setting validated the effect of CORTEX on decision effectiveness and its side-effects on product quality and execution efficiency. Data was collected in a survey with both numerical and open-ended questions.

The case study was primarily qualitative in nature and not strictly controlled. Qualitative research rarely has the chance to properly mitigate and plan all possible threats to validity and mostly focuses on solving these issues during the study itself [147, 27]. Even though the project was set up in an intrapreneurship environment, it still maintained a realistic project environment to evaluate the application of CORTEX. This meant to deal with standard issues like politics, status reporting, uncertainty in product development, limited time and budget as well as limited accessibility to customers and software systems.

Construct Validity Validation tested CORTEX in different configurations via the simulation and additionally employed a case study. This reduces the risk of a mono-operation as well as mono-method bias. In the case study, subjects could have guessed the purpose of the research and altered their behavior. It was therefore communicated as an innovation project without emphasizing the process. Hypothesis guessing was not a factor in the simulation. In addition, subjects of the case study could have felt pressure while being evaluated, causing them to give favorable answers. Evaluation apprehension was not a factor in the simulation.

Internal Validity Temporal precedence was no factor for the simulation since it only involved executing CORTEX. Since the case study was restricted to one independent variable with a single-subject design and linear execution, there is no ambiguity about the temporal precedence. For the same reason, there is no threat of diffusion, i.e., treatment effects spreading to a control group. Potential confounding variables are ability or willingness to learn a new process as well as intuitiveness in unguided decision-making. Characteristics of project participants were not controlled for. However, the survey targeted both participants of the development process as well as outside stakeholders. Since the project was staffed independently from the case study, there is no bias through self-selection. The survey captured cross-influences between design, functionality, and performance to assess the outcome of decisions. The project was staffed with experienced practitioners and the validation period was not long-term. Therefore there is no risk of bias through maturation, i.e., the experience level of participants rising significantly on its own. Case study participants might have exited the project out of frustration with CORTEX. However, no such events were recorded by the organization. The simulation was neither affected by maturation nor by survivorship bias.

External Validity Generalizability could have been impacted by a study design not representative of typical CSE projects. For the simulation, any type of modeling risks loss

of detail through simplification and abstraction. Additionally, simulations are software which carries the risk of bugs. Calibration, verification, and validation ensured an accurate model. The case study subject may not have been representative of the target population, introducing selection bias. This is especially the case with a single subject and might have been exacerbate by convenience sampling. To mitigate selection bias and subjectiveness of metrics, the survey targeted participants from different stakeholder groups. Selection bias is not a factor for the simulation.

Reliability With only one researcher interpreting the results, the study may have suffered from confirmation bias. Case study execution and interpretation of results were peer reviewed at the chair. Additionally, questions using a numerical scale were designed to leave little room for interpretation When interpreting answers to free text questions, the line of reasoning was explained. As a characteristic of technical action research, the researcher could have also influenced participants to behave according to expectations. Therefore, an intermediary interacted with the project.

5.2.3 RADAR Validation

RADAR validation focused on the benefits of RADAR for decision support and knowledge management when applying CORTEX. Technical action research used an industry case study to establish CORTEX supported by RADAR. Data was collected using a survey as well as from the knowledge management and project management subsystems.

Construct Validity Validation only tested a single, prototypical implementation of RADAR in a single study design which could imply a mono-operation and mono-method bias. The methods for knowledge management and decision support applied by RADAR are based on literature review as well as references from the tool survey, suggesting that this study design is sufficient. In the case study, subjects could have guessed the purpose of the research and altered their behavior. It was therefore communicated as an innovation project without emphasizing the process. In addition, subjects of the case study could have felt pressure while being evaluated, causing them to give favorable answers. The survey was strengthened with quantitative data.

Internal Validity Since the case study was restricted to one independent variable with a single-subject design and linear execution, there is no ambiguity about the temporal precedence. For the same reason, there is no threat of diffusion, i.e., treatment effects spreading to a control group. Potential confounding variables are ability or willingness to learn a new process as well as intuitiveness in unguided decision-making. Characteristics of project participants were not controlled for. However, the survey targeted both participants of the development process as well as outside stakeholders. Since the project was staffed independently from the case study, there is no bias through self-selection. The project was staffed with experienced practitioners and the validation period was not long-term. Therefore there is no risk of bias through maturation, i.e., the experience level of participants rising significantly on its own. Case study participants might have exited the project out of frustration with CORTEX and RADAR. However, no such events were recorded by the organization.

External Validity Generalizability could have been impacted by a study design not representative of typical CSE projects. The case study subject may not have been representative of the target population, introducing selection bias. This is especially the case with a single subject and might have been exacerbate by convenience sampling. To mitigate selection bias and subjectiveness of metrics, the survey targeted participants from different stakeholder groups.

Reliability With only one researcher interpreting the results, the study may have suffered from confirmation bias [148]. Case study execution and interpretation of results were peer reviewed at the chair. Additionally, questions using a numerical scale were designed to leave little room for interpretation When interpreting answers to free text questions, the line of reasoning was explained. As a characteristic of technical action research, the

researcher could have also influenced participants to behave according to expectations. Therefore, an intermediary interacted with the project.

5.3 Future Work

The design science research project yielded the artifacts CRDM, CORTEX, and RADAR. Each artifact requires further validation as well as evaluation to strengthen results on their reliability, flexibility, and efficacy. Additionally, there are possibilities for expanding both CORTEX and RADAR for future applications.

Validation and Evaluation Validation of all artifacts should be furthered to address the threats to validity. First and foremost, controlled experiments could be used to control for confounding variables that might have impacted past results. Additionally, past validation need to be reproduced to rule out confirmation bias. The software simulation of CORTEX could also be used to analyze the sensitivity of the process framework to parameters such as prioritization mechanism, risk affinity, and project capacity. Benchmarking with related process models could be performed efficiently using simulation as well. Since the research project did not cover implementation and evaluation according to Wieringa [63], both CORTEX and RADAR should be implemented in one or multiple real CSE organization and evaluated in long-term usage. Even though validation used technical action research to test artifacts in a real-world context, evaluation would allow deeper insights into their effects and limitations.

Expansion and Extension Expansion means adding new details to the existing mechanisms. CORTEX should be expanded to cover more edge cases: Research failed and there is no result, implementation showed that solution estimation was wrong, additional facts such as problem benefit are actively reported. On the other hand, extension means adding new concepts and functionality. Both CORTEX and RADAR could be extended with additional features: Goals could be set more differentiated than “solve this problem”. Progress of implementation should be tracked since solution are not implemented in one step and problems do not suddenly change from being unsolved to being solved. Additionally, as already explored in RADAR validation, decision support could benefit from automations to determine need for research and opportunities for implementation.

Appendices

Appendix A

Licenses

Appendix B

CRDM

B.1 Design of CRDM

B.1.1 Functional Model

The following tables describe the use cases of CRDM's functional model in detail by specifying preconditions, participating roles, flow of events, and postconditions.

Use case	**Problem Solving: Capture Problem**
Participating roles	R&D Manager, Customer, Domain Expert
Preconditions	• Customer feels a need that is not currently captured in the knowledge base. • The need has been recognized through customer report or research activity.
Flow of events	1. Customer describes domain problem to be solved. 2. R&D manager captures problem in knowledge base and connects it to existing entities. 3. Domain experts provide further input such as problem details, dependencies, sub-problems, value, potential solutions. 4. R&D manager captures additional information in knowledge base. 5. R&D manager provides feedback to customer about captured problem.
Postconditions	• Problem has been captured including all information that is initially available. • Customer feels that their need has been captured appropriately.

Table B.1: Description of use case 1, capturing a problem by R&D manager, customer, and domain experts as part of problem solving.

Use case	**Problem Solving: Deliver Solution**
Participating roles	R&D Manager, Customer
Preconditions	• Information on problem and solution space is captured in the knowledge base. • The Continuous R&D organization has capacity to work on solutions.
Flow of events	1. R&D manager reviews knowledge base for opportunities to solve valuable problems. 2. R&D manager reviews knowledge base for opportunities to utilize economical solutions. 3. R&D manager weighs risk of implementing a particular solution for a particular problem. 4. R&D manager either decides on implementing a solution or postpones decision. 5. R&D manager commissions implementation tasks and provides details on goal and priority. 6. R&D manager oversees implementation of the solution and delivery to customer.
Postconditions	• Customer received solutions for valuable problems. • Costs/benefit ratio of implementation activities was maximized. • High-risk decisions were postponed.

Table B.2: Description of use case 2, guiding the delivery a solution by R&D manager as part of problem solving.

Use case	**Problem Solving: Capture Feedback**
Participating roles	R&D Manager, Customer, Domain Expert
Preconditions	• Information on problem and solution space is captured in the knowledge base.
Flow of events	1. R&D manager collects feedback to validate correctness, completeness, consistency, clarity of knowledge base contents. 2. Customer provides feedback on solution selected for implementation, e.g., regarding priority of the underlying problem and suitability of the solution. 3. Domain experts provide feedback on the planned or ongoing implementation, e.g., regarding feasibility and anticipated effects. 4. Customer provides feedback on delivered solution, e.g., regarding usability, effectiveness, and potential follow-up problems.
Postconditions	• Knowledge base is updated and extended with additional insights gained during research and development.

Table B.3: Description of use case 3, capturing feedback by R&D manager, customer, and domain experts as part of problem solving.

Use case	**Decision Support: Assess Uncertainty**
Participating roles	R&D Manager, Analyst
Preconditions	• Information on problem and solution space is captured in the knowledge base.
Flow of events	1. R&D manager identifies areas where uncertainty in problem or solution domain has impact on decision-making. 2. Analysts assess each area with respect to locations, types, and degrees of uncertainty. 3. Analysts rate confidence in knowledge base contents based on inherent uncertainty. 4. R&D manager captures information on uncertainty and corresponding confidence in the knowledge base.
Postconditions	• Information on uncertainty and corresponding confidence in knowledge base contents is captured in the knowledge base.

Table B.4: Description of use case 4, assessing uncertainty in the knowledge base by R&D manager and analysts as part of decision support.

Use case	**Decision Support: Reduce Uncertainty**
Participating roles	R&D Manager, Analyst
Preconditions	• Information on problem and solution space is captured in the knowledge base. • Uncertainty in knowledge base has been assessed.
Flow of events	1. R&D manager reviews knowledge base regarding potential risks involved in upcoming decisions. 2. R&D manager identifies areas with unacceptable risk/reward ratio. 3. Analysts devise research strategy and methodology to reduce risk. 4. Analysts describe and prioritize research task to conduct the research. 5. Analysts execute research task according to the organization's execution process model. 6. Analysts collect and evaluate results of the research task. 7. Analysts update knowledge base with insights gained from research. 8. R&D manager reviews updated knowledge base and reassesses risk/reward situation. 9. If necessary, R&D manager commissions continued research to further reduce uncertainty.
Postconditions	• Knowledge base carries more information and less uncertainty in areas that are critical for upcoming decisions.

Table B.5: Description of use case 5, reducing uncertainty in the knowledge base by R&D manager and analysts as part of decision support.

Use case	**Decision Support: Find Opportunities**
Participating roles	R&D Manager, Analyst
Preconditions	• Information on problem and solution space is captured in the knowledge base. • Uncertainty in knowledge base has been assessed and reduced to an acceptable level.
Flow of events	1. R&D manager recognizes a change in the knowledge base that might imply an opportunity for problem solving. 2. R&D manager reviews cost/benefit ratio as well as risk/reward ratio of the opportunity. 3. If necessary, analysts conduct additional research to validate the opportunity. 4. R&D manager decides whether or not to seize the opportunity, possibly changing the organization's previous plans for implementation. 5. If necessary, R&D manager confers with stakeholders about the decision. 6. R&D manager describes and prioritizes implementation task to seize the opportunity.
Postconditions	• Opportunity is captured by an implementation task in the organization's backlog.

Table B.6: Description of use case 6, discovery of opportunities for problem solving by R&D manager and analysts as part of decision support.

Use case	**Continuous Execution: Discover Problems**
Participating roles	Analyst, Domain Expert
Preconditions	• Stakeholder reports problem for organization to work on. • OR Organization has capacity to work on additional problems.
Flow of events	1. Analysts investigate problem space, looking for additional problems or following up on reported problems. 2. Analysts and domain experts devise suitable research activities to increase information and reduce uncertainty. 3. Analysts execute research activities and evaluate results regarding existing problems. 4. Analysts and domain experts investigate discovered problems regarding structure, implications, value, connections, and uncertainty. 5. Analysts and domain experts break down problems into smaller parts, depending on scope, structure, complexity, and connected solutions. 6. Analysts capture problems and available information in the knowledge base. 7. Analysts provide feedback about captured problems to R&D manager and stakeholders.
Postconditions	• Problems are captured in the knowledge base along with available information. • R&D manager and stakeholders are informed about captured problems.

Table B.7: Description of use case 7, discovery of problems by analysts and domain experts as part of continuous execution.

Use case	**Continuous Execution: Discover Solutions**
Participating roles	Analyst, Domain Expert, Developer
Preconditions	• Stakeholder or member reports solution for organization to utilize. • OR Organization decides to investigate solution options for known problem.
Flow of events	1. Analysts, domain experts, and developers investigate solution space, looking for additional solutions or following up on reported solutions. 2. Analysts, domain experts, and developers devise suitable research activities to increase information and reduce uncertainty. 3. Analysts execute research activities and evaluate results regarding available solutions. 4. Analysts and domain experts investigate discovered solutions regarding structure, implications, cost, connections, and uncertainty. 5. Analysts and domain experts break down solutions into smaller parts, depending on scope, structure, complexity, and connected problems. 6. Analysts capture solutions and available information in the knowledge base. 7. Analysts provide feedback about captured solutions to R&D manager and stakeholders.
Postconditions	• Solutions are captured in the knowledge base along with available information. • R&D manager and stakeholders are informed about captured solutions.

Table B.8: Description of use case 8, discovery of solutions by analysts and domain experts as part of continuous execution.

Use case	**Continuous Execution: Prioritize Problems**
Participating roles	Analyst, Domain Expert
Preconditions	• Organization has defined a mechanism for benefit estimation. • Organization has defined a mechanism for problem prioritization. • Problems have been captured in the knowledge base. • R&D manager requires prioritization of problems for decision support.
Flow of events	1. Analysts and domain experts investigate problems in knowledge base and assess available information as well as uncertainty regarding benefit values. 2. Analysts and domain experts devise suitable research activities to increase information and reduce uncertainty. 3. Analysts execute research activities and evaluate results regarding problem benefit and priority. 4. Analysts update estimated benefit values as well as inherent uncertainty in knowledge base. 5. Analysts update prioritization of problems and connected solutions according to new information. 6. Analysts provide feedback about updated estimates and prioritization to R&D manager and stakeholders.
Postconditions	• Estimates for problem benefit values along with inherent uncertainty are captured in the knowledge base. • Prioritization of problems and connected solutions is updated in knowledge base. • R&D manager and stakeholders are informed about captured estimates and priorities.

Table B.9: Description of use case 9, prioritization of problems by analysts and domain experts as part of continuous execution.

Use case	**Continuous Execution: Estimate Solutions**
Participating roles	Analyst, Domain Expert, Developer
Preconditions	• Organization has defined a mechanism for cost estimation. • Solutions have been captured in the knowledge base. • R&D manager requires estimation of solutions for decision support.
Flow of events	1. Analysts, domain experts, and developers investigate solutions in knowledge base and assess available information as well as uncertainty regarding cost values. 2. Analysts, domain experts, and developers devise suitable research activities to increase information and reduce uncertainty. 3. Analysts and developers execute research activities and evaluate results regarding solution costs. 4. Analysts update estimated cost values as well as inherent uncertainty in knowledge base. 5. Analysts update prioritization of problems and connected solutions according to new information. 6. Analysts provide feedback about updated estimates and prioritization to R&D manager and stakeholders.
Postconditions	• Estimates for solution cost values along with inherent uncertainty are captured in the knowledge base. • Prioritization of problems and connected solutions is updated in knowledge base. • R&D manager and stakeholders are informed about captured estimates and priorities.

Table B.10: Description of use case 10, estimation of solutions by analysts and domain experts as part of continuous execution.

Use case	**Continuous Execution: Evaluate Suitability**
Participating roles	Analyst, Domain Expert, Developer
Preconditions	• Problems and solutions have been captured in the knowledge base. • Organization has defined a mechanism for suitability evaluation. • R&D manager requires evaluation of problem/solution suitability for decision support.
Flow of events	1. Analysts, domain experts, and developers investigate problems and solutions in knowledge base and assess available information as well as uncertainty regarding suitability. 2. Analysts, domain experts, and developers devise suitable research activities to increase information and reduce uncertainty. 3. Analysts and developers execute research activities and evaluate results regarding problem/solution suitability. 4. Analysts update estimated suitability values as well as inherent uncertainty in knowledge base. 5. Analysts update prioritization of problems and connected solutions according to new information. 6. Analysts provide feedback about updated suitabilities and prioritization to R&D manager and stakeholders.
Postconditions	• Evaluated suitabilities of solutions for problems along with inherent uncertainty are captured in the knowledge base. • Prioritization of problems and connected solutions is updated in knowledge base. • R&D manager and stakeholders are informed about captured estimates and priorities.

Table B.11: Description of use case 11, evaluation of problem/solution suitability by analysts and domain experts as part of continuous execution.

Use case	**Continuous Execution: Implement Solution**
Participating roles	Developer, Analyst, (R&D manager)
Preconditions	• Organization has defined a model for implementation execution. • Organization has defined a model for solution delivery. • Opportunities for problem solving have been discovered.
Flow of events	1. Developers analyze and refine implementation tasks required to implement the chosen solution to the chosen problem, consulting any other organization member or stakeholders if necessary. 2. Developers organize implementation tasks according to the organization's execution process model, e.g., in a product backlog (Scrum) or a task queue (Kanban). 3. Developers execute implementation tasks according to the organization's execution process model, e.g., based on the sprint backlog (Scrum) or the current free capacity (Kanban). 4. Developers deliver results of implementation according to the organization's delivery model, e.g., to a staging environment for release by the R&D manager (continuous delivery) or immediately to production (continuous deployment). 5. Developers notify organization about results of the implementation activity.
Postconditions	• Progress is made on the solution of selected problems. • Created or updated solution is delivered to customers. • Organization members and stakeholders are informed about progress on the solution.

Table B.12: Description of use case 12, implementation of solutions by analysts and domain experts as part of continuous execution.

B.1.2 Object Model

Variability Management Pohl et al. [96] describe the orthogonal variability modeling approach (OVM) integrated in CRDM's object model. The following lists contain important definitions from OVM that are relevant to the application in CRDM. **Managing variability** spans activities such as supporting, defining, and exploiting variability. **Defining variability** is the sum of all activities concerned with the identification and documentation of variability **Exploiting variability** refers to binding variants during application engineering.

Variability management is carried out by managing variable artifacts and trace information needed to fulfill these activities. **Development artifact**s are defined as "the output of a sub-process of domain or application engineering." Examples for development artifacts are requirements, architecture, components, and tests." **Domain artifact**s are "reusable development artifacts created in the sub-processes of domain engineering." **Application artifact**s are "development artifacts of specific product line applications."

Variability is differentiated into internal and external variability. **External variability** stems from domain artifacts that are visible to customers. **Internal variability** stems from domain artifacts that are hidden from customers. Variability concerns certain items or properties in the real world. **Variability subject**s are variable items or properties in general. **Variability object**s are particular instances of variability subjects. Within a given context, **variation point**s represent variability subjects within domain artifacts. **Variant**s represent variability objects within domain artifacts.

B.2 Validation of CRDM Applicability

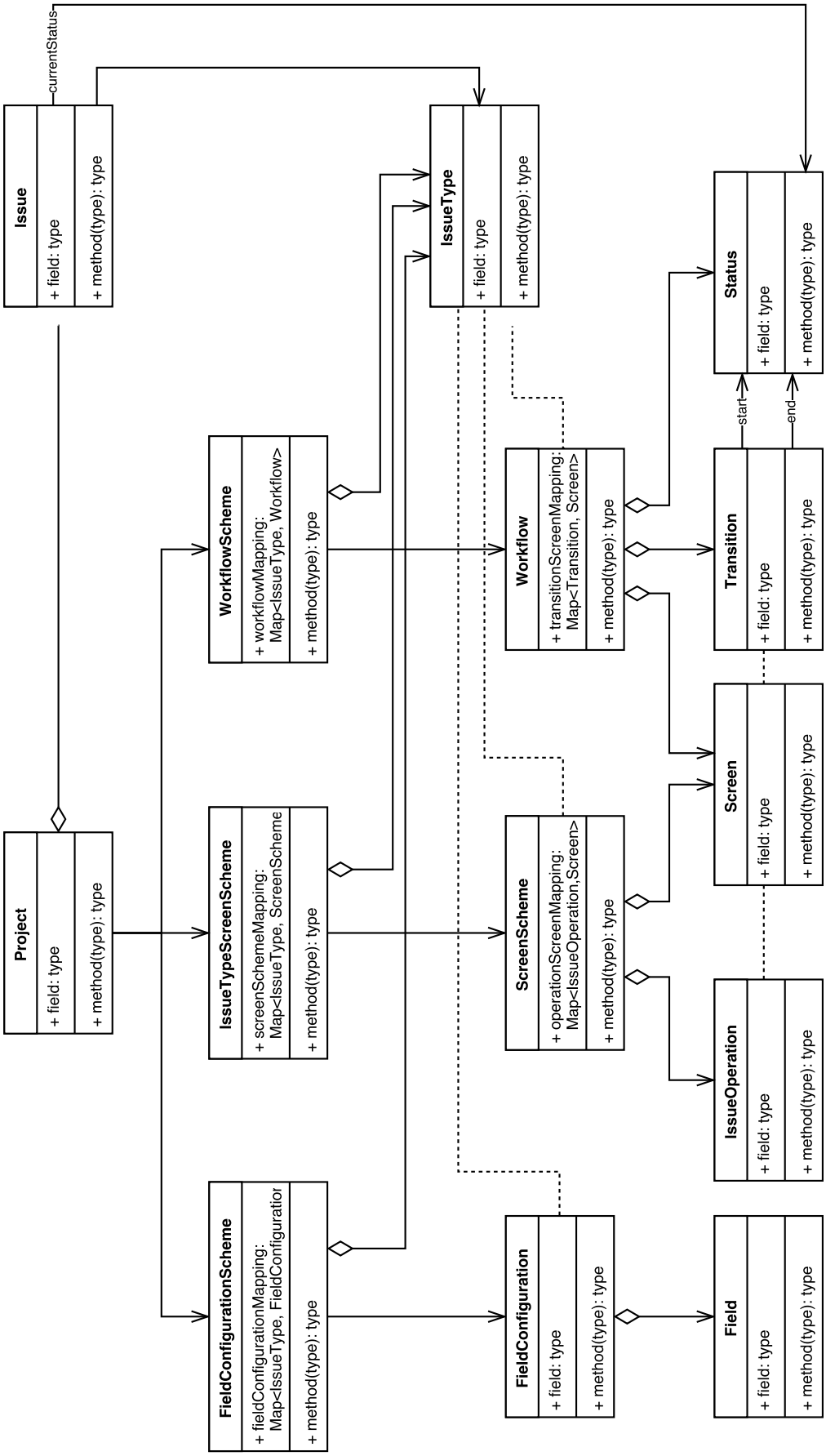


Figure B.1: Jira object model covering work items, workflow, and visual representations; adapted from [149].

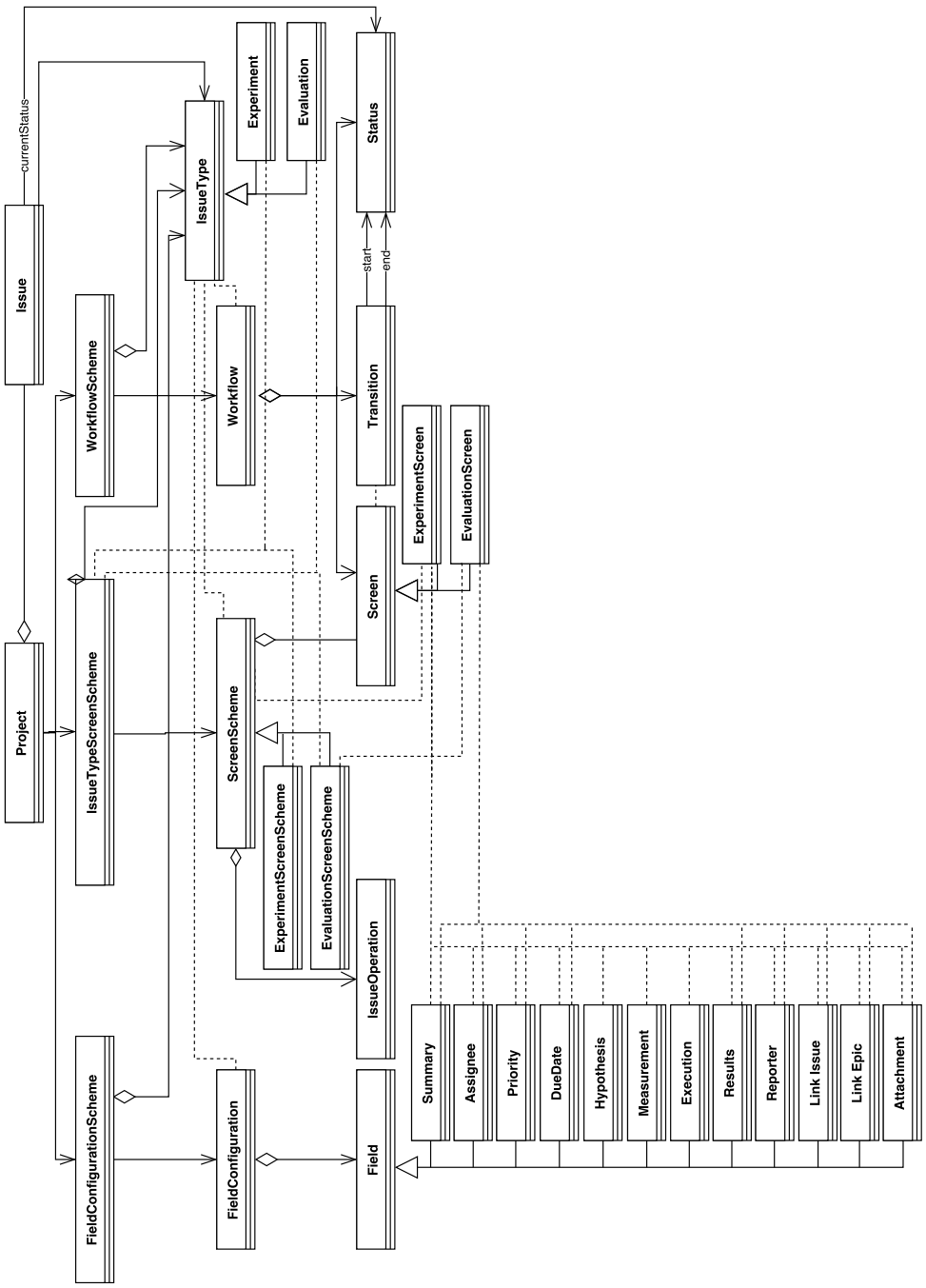


Figure B.2: Jira object model extended with custom issue types, fields, screen schemes, and screens.

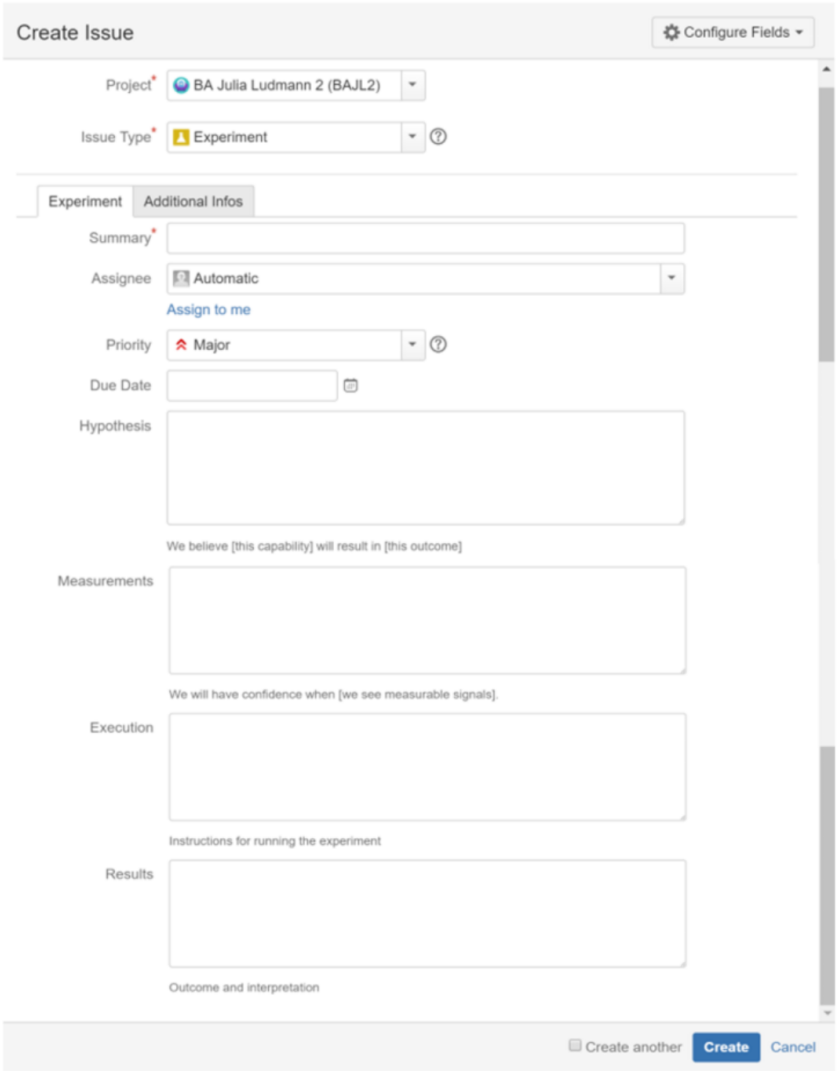


Figure B.3: Experiment ticket to record hypotheses, set up the structure and record the results.

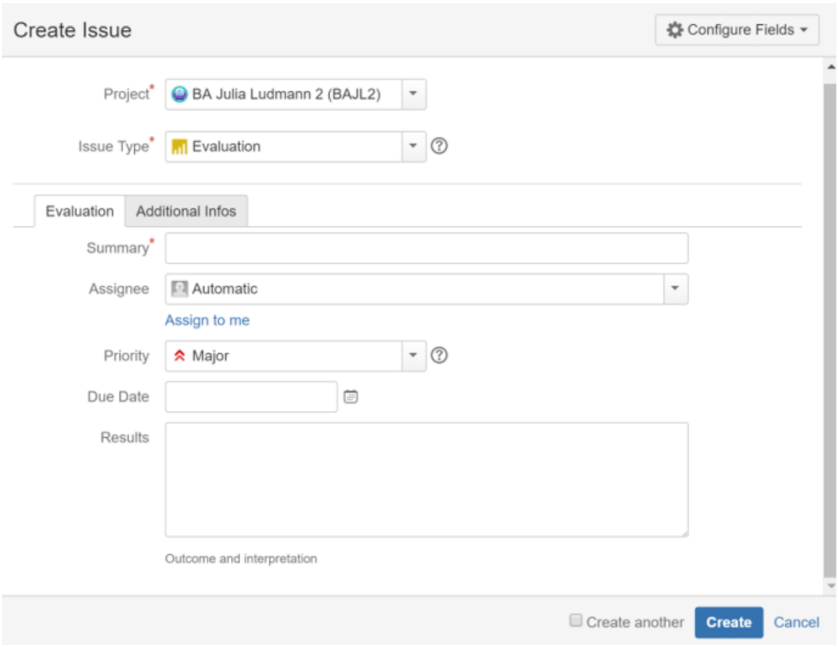


Figure B.4: Evaluation ticket for the subsequent evaluation of long-term experiments.

Appendix C
CORTEX

C.1 Gap Analysis of Related Process Models

Category	HDE	IES	ESSSDM	EVAP	HYPEX	QCD	DVOCE	RIGHT
Prediction	1	0	1	1	1	2	1	0
Continuity	0	1	0	0	1	2	1	2
Experimentation	0	0	0	0	1	0	1	2
Opportunism	0	0	1	1	0	0	1	0
Refinement	0	0	0	0	1	1	0	0
Innovation	2	1	2	0	1	1	0	0
Decision-making	2	0	1	1	1	1	1	0
Research methodology	0	2	1	0	0	1	1	0

Table C.1: Rating of CSE process models for continuous innovation and experimentation in relevant categories.

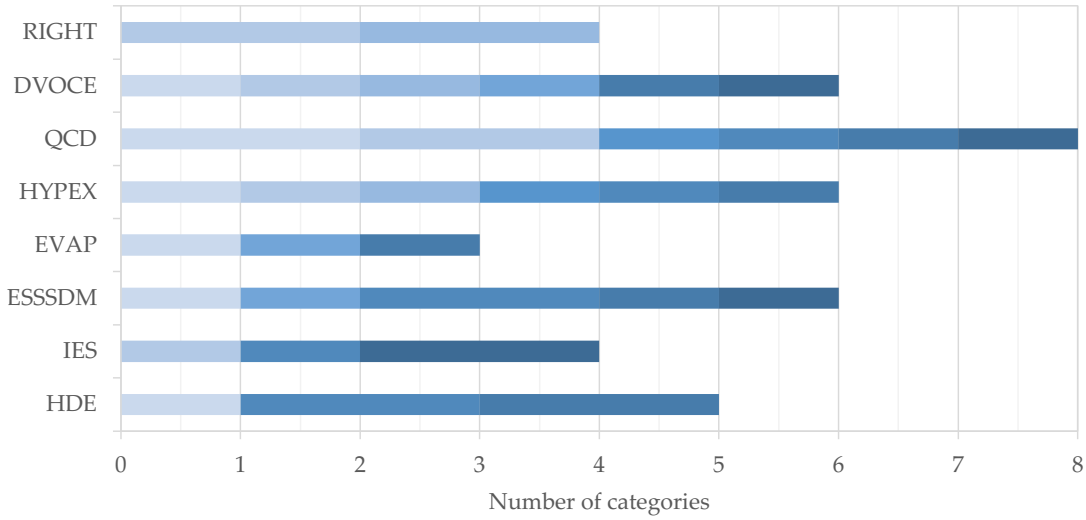


Figure C.1: Number of categories covered by each process model.

C.2 Design of CORTEX Process Framework

C.2.1 Research Techniques

The following compilation is based on preliminary works [123, 61, 124].

Research Strategies

Research strategy	Description
Sequential explanatory	• Quantitative data followed by qualitative data • Useful when unexpected results arise from the quantitative phase • Example: Controlled Experiment followed by case study
Sequential exploratory	• Qualitative data followed by quantitative data • Useful for testing elements of an emerging theory • Example: Ethnography followed by survey
Concurrent triangulation	• Concurrent use of different research methods to confirm, cross-validate or corroborate findings • Multiple sources to strengthen validity and mitigate weaknesses • Problem: Comparing results and resolving contradictions may be difficult and a larger amount of data needs to be processed

Table C.2: Approaches to research strategies as identified by Creswell [103] and Easterbrook et al. [104].

Research Methods

Wieringa [63] reduce methods to study validation models to the following list. These so-called *analogical* models are easier to understand than the actual implementation in

the actual context. Such models can be mock-ups, physical or digital prototypes, real stakeholders or stand-ins, software simulations, artificial environments in a laboratory, etc.

Expert opinions submits an artifact to a panel of experts who imagine the effects of applying it in the problem context. They can be very efficient but require careful design.

Single-case mechanism experiments apply a single stimulus to the model (e.g., a prototype) and try to explain the response based on the knowledge about mechanisms within the model.

Statistical difference-making experiments compare the average outcome of applying the treatment to samples and are useful if mechanisms are not fully understood but require effort to control conditions.

Technical action research (TAR) uses a representative real-world case as a model for the entire problem context and tries to derive learnings, usually as part of scaling up from the laboratory to the real world.

Easterbrook et al. [104] identifies and describes five classes of empirical research methods. These classes can also be combined in a mixed-method approach, which is powerful but also very time-intensive to employ and leads to a large volume of information to be processed. Mixed strategies often evaluate evidence from both qualitative and quantitative research to validate theories. Through this, limitations of individual methods are compensated by strengths found within other methods [104].

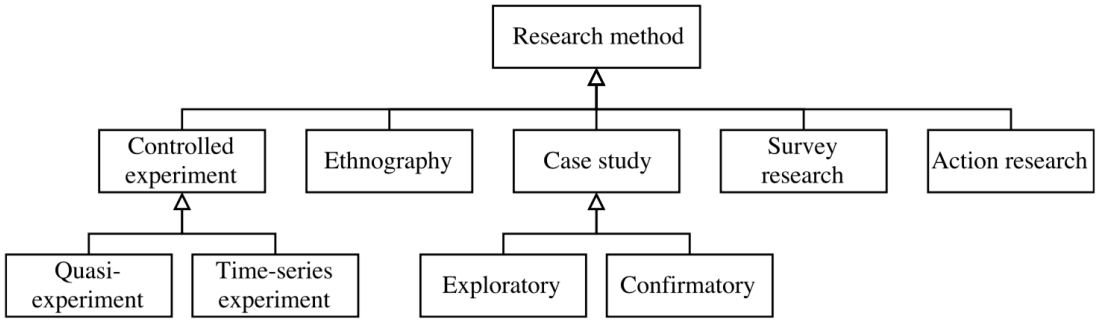


Figure C.2: Empirical research method classes (derived from [104]).

Controlled Experiments aim to investigate testable, clearly formulated hypotheses by manipulating and combining a set of independent variables. The impact of each combination, also called treatment, is then measured on dependent variables to find a cause-effect relationship. As variants, quasi-experiments, where the user can choose which treatment they want to evaluate and time-series experiments, where effects are measured stepwise over a certain period of time, can be employed.

Weaknesses of controlled experimental approaches are that correlations may occur by chance ("fishing for results"), or that effects of important variables are ignored by mistake [104].

Case Study is described as "an empirical inquiry that investigates a contemporary phenomenon within its real life context, [...]" [150, 104]. It can be split into two main categories: Exploratory case studies serve as an"initial investigation of some phenomena to derive new hypotheses and build theories." [104] while confirmatory case studies aim to test or compare existing theories within a real application environment. Compared to controlled experiments, it mitigates weaknesses where a realistic context is required or an effect can only be measured over a long-term application. The results however are often open to researcher interpretation and potentially subject to bias [104].

Survey Research uses a data collection method (e.g. questionnaire) on a selected representative sample group from a well-defined population. The results are generalized using data analysis techniques. Clearly formulated research questions and a properly sampled set of representatives are required to avoid bias. Sampling bias, low response rate or non-valuable data due to question design errors may still weaken validity and generalizability of results [104].

Ethnography means conducting a study on a community of people regarding their social behavior. This can be achieved for example through participant observation, where a researcher becomes part of the community over a period of time to gain insider knowledge. The method tends to be very time intensive and difficult to employ in terms of correct observations and data evaluation, but yields deep insight when performed correctly [104].

Action Research implements a solution and studies its impact on a real-world problem. As an empirical method, action research is still immaturely studied, leading to vague evaluation frameworks. Due to its practical nature it enables passing on and reflecting experiences and learning outcomes [104]).

C.2.2 Prioritization and Estimation

The following compilation is based on preliminary works [123, 139, 61, 124].

Selection of techniques (not exhaustive) for solving estimation and prioritization problems that are often used in software development projects.

Analytical Hierarchy Process (AHP) Compare all possible pairs of alternatives to determine a prioritization. AHP is not suitable for a large number of objects ($\frac{n \cdot (n-1)}{2}$ comparisons), but for instance identifies redundancy or inconsistencies between requirements [75].

Cumulative Voting or the 100 Dollar Test Distribute imaginary units adding up to a certain number between requirements. Stakeholders presented with this technique may prioritize arbitrarily or strategically, thus giving heavy weight to one requirement they personally favor while not balancing out evenly with other requirements [75].

Numerical Assigning (Grouping) Assign a category of critical, standard or optional to requirements. Since "everything is critical" to the customer, this may not lead to a beneficial classification [75].

Ranking Assign a tie-free, numerical ranking to a set of requirements. If different rankings are performed for multiple stakeholders, it may be difficult to merge the results into an overall ranking [75].

Top-Ten Requirements Pick an unordered list of the ten most important requirements. Stakeholders may feel left out when their requirement does not appear on the list, thus it is important to evenly balance out the list and select requirements from many areas [75].

Eisenhower's Principle Categorize alternatives according to their importance-urgency estimations. Importance usually refers to achieving a certain key goal or outcome, while urgent tasks demand immediate attention since otherwise heavy consequences may follow. Usually, four quadrants are employed for categorizing items, each yielding one of the following descriptions: important and urgent, important but not urgent, not important but urgent and finally neither important nor urgent. By locating and weighing items within these quadrants, the prioritization can be graphically visualized.[1]

RICE Model Apply a scoring system for prioritization by evaluating the factors reach, impact, confidence and effort.

- Reach: Number of involved customers, possibly factored with frequency of software invocations.

[1] MindTools. *Eisenhower's Urgent/Important Principle*. 2016. URL: `https://www.mindtools.com/pages/article/newHTE_91.htm`.

- Impact: How much some functionality will affect customers.
- Confidence: How much confidence is placed in the correctness of assumptions.
- Effort: How many working hours will be spent on involved tasks.

These factors are combined into a single score: $RICE = \frac{Reach \times Impact \times Confidence}{Effort}$ If many individual scores are calculated with low confidence, they may be misleading unless the overall confidence is increased.[2]

RICC Model Variation of the RICE model whereby effort (E) is replaced with complexity (C) to express the uncertain amount of time and resources required by a solution instead of attempting to quantify the resulting effort.[3] The factors are combined in the same way as with RICE: $RICC = \frac{Reach \times Impact \times Confidence}{Complexity}$

Kano Model Classify requirements by assigning them to one out of five categories. It measures expected customer satisfaction by defining excitement, performance, basis, indifferent or reverse requirements. Since the classification in this context is subjective, it needs to be validated with stakeholders, e.g., by means of a structured questionnaire or interviews[4].

Problem Occurence A possible way to measure importance of a problem, coming from usability research, is to measure the impact, persistence, and frequency of the problem [153]:

Impact How much trouble will affected users experience?

Persistence How many times will a user experience the problem?

Frequency How many users will be affected by the problem?

These three factors are measured individually, but the result can be a combination of all three factors. The rating of the individual factors can be summed up or multiplied to yield a combined score [154].

Customer Feedback Results can be further strengthened through additional testing and customer feedback techniques [35]. With action research, deploying some form of evolutionary or incremental prototype[5] or even a live product enables continuous observation of feature use. If an interview approach is chosen to evaluate solution preferences, a throwaway prototype or mockup of solution variants may be sufficient [156].

[2] S. McBride. *RICE: Simple prioritization for product managers*. 2016. URL: https://www.intercom.com/blog/rice-simple-prioritization-for-product-managers/.

[3] S. McBride. *RICE: Simple prioritization for product managers*. 2016. URL: https://www.intercom.com/blog/rice-simple-prioritization-for-product-managers/.

[4] Kano Model. URL: https://www.kanomodel.com/

[5] Scottish Qualifications Authority (SQA). *F1VT 34: Interactive Media: Authoring: Creating a screen-based prototype*. 2007. URL: https://www.sqa.org.uk/e-learning/IMAuthoring01CD/index.htm.

Numerical/Nominal Sizing Cost of solutions can be estimated it in different ways, ranging from mathematical criteria [157] to evaluating the cost on a nominal scale (e.g., "t-shirt sizes") [158, 159]. The resulting values may be combined with an uncertainty rating to express how confident the team is about the estimation.

C.3 Validation of CORTEX Integrity

Listing C.1: `DecisionMaking` module of CORTEX simulation, implemented in Swift.

```
func generateNextTasks(knowledge: Knowledge, pipeline: TaskPipeline, minExpectedROI: Double, params: SimulationParameters) -> Set<Task> {
    let tackledProblems = pipeline.implementationTasks
        .compactMap { knowledge.getSolution($0.solutionUUID) }
        .compactMap { knowledge.problemFor($0) }
    let untackledProblems = knowledge.problems.subtracting(tackledProblems)
    let optimalChoices = untackledProblems.compactMap { chooseOptimalSolution(for: $0, with: knowledge, minExpectedROI: minExpectedROI) }
    let implementationTasks = optimalChoices.map { ImplementationTask(for: $0.solution) }
    let problemsWithDecision = optimalChoices.map { $0.problem }
    let problemsWithoutDecision: Set<Problem> = untackledProblems.subtracting(problemsWithDecision)
    let researchTasks = problemsWithoutDecision.flatMap { determineResearchTasks(for: $0, knowledge: knowledge, pipeline: pipeline, params: params) }
    return Set(implementationTasks).union(researchTasks)
}

func chooseOptimalSolution(for problem: Problem, with knowledge: Knowledge, minExpectedROI: Double) -> KnowledgeChain? {
    let minExpectedROIFulfilled: (KnowledgeChain) -> Bool = {
        expectedROI(problem: problem, connection: $0.connection, solution: $0.solution) >= minExpectedROI
    }
    let greaterExpectedROI: (KnowledgeChain, KnowledgeChain) -> Bool = { lhs, rhs in
        let lhsReturn = expectedROI(problem: problem, connection: lhs.connection, solution: lhs.solution)
        let rhsReturn = expectedROI(problem: problem, connection: rhs.connection, solution: rhs.solution)
        return lhsReturn > rhsReturn
    }
    return knowledge.connectedSolutions(for: problem)
        .sorted(by: greaterExpectedROI)
        .first(where: minExpectedROIFulfilled)
}

func expectedROI(problem: Problem, connection: Connection, solution: Solution) -> Double {
    let roi = benefitCostRatio(problem: problem, solution: solution) - 1
    let confidence = overallConfidence(problem: problem, connection: connection, solution: solution)
    return (confidence.value > 0)
        ? confidence * roi
        : 0
}

func overallConfidence(problem: Problem, connection: Connection, solution: Solution) -> Confidence {
    return problem.confidence * problem.benefit.confidence * connection.confidence * solution.confidence * solution.cost.confidence
}
```

```
39
40  func benefitCostRatio(problem: Problem, solution: Solution) ->
        Double {
41      return (solution.cost.value > 0)
42          ? problem.benefit / solution.cost
43          : Double.infinity
44  }
45
46  func determineResearchTasks(for problem: Problem, knowledge:
        Knowledge, pipeline: TaskPipeline, params: SimulationParameters)
        -> Set<ResearchTask> {
47      let knowledgeChains = knowledge.connectedSolutions(for: problem)
48      let problemTask = determineResearchForProblem(problem: problem,
        threshold: params.minConfidence)
49      let solutionTasks = knowledgeChains.map {
        determineResearchForSolution(solution: $0.solution,
        minConfidence: params.minConfidence) }
50      let connectionTasks = knowledgeChains.map {
        determineResearchForConnection(connection: $0.connection,
        minConfidence: params.minConfidence) }
51      let tasks = ([problemTask] + solutionTasks +
        connectionTasks).compactMap { $0 }
52      let pendingResearchTasks = pipeline.pendingResearchTasks
53      let redundantTasks = tasks.filter { task in
        pendingResearchTasks.contains(where: { pendingTask in
        task.redundant(pendingTask) }) }
54      return Set(tasks).subtracting(redundantTasks)
55  }
56
57  func determineResearchForProblem(problem: Problem, threshold:
        Confidence) -> ResearchTask? {
58      return (problem.confidence < threshold ||
        problem.benefit.confidence < threshold)
59          ? EstimateProblemTask(problem)
60          : nil
61  }
62
63  func determineResearchForConnection(connection: Connection,
        minConfidence: Confidence) -> ResearchTask? {
64      return (connection.confidence < minConfidence)
65          ? EstimateConnectionTask(connection)
66          : nil
67  }
68
69  func determineResearchForSolution(solution: Solution,
        minConfidence: Confidence) -> ResearchTask? {
70      return (solution.confidence < minConfidence ||
        solution.cost.confidence < minConfidence)
71          ? EstimateSolutionTask(solution)
72          : nil
73  }
```

Listing C.2: `Entity` module of CORTEX simulation, implemented in Swift.

```
protocol Fact: Equatable {
    var confidence: Confidence { get }
}

class Entity: Fact {
    let uuid: UUID
    let confidence: Confidence

    init(uuid: UUID = UUID(), with confidence: Confidence) {
        self.uuid = uuid
        self.confidence = confidence
    }
}

extension Entity: Hashable {
    func hash(into hasher: inout Hasher) {
        hasher.combine(uuid)
    }
}

extension Entity: Equatable {
    static func == (lhs: Entity, rhs: Entity) -> Bool {
        lhs.hashValue == rhs.hashValue
            && lhs.confidence == rhs.confidence
    }
}

struct Confidence: Hashable {
    let value: Double

    init(_ confidence: Double) {
        self.value = min(max(confidence, 0), 1)
    }

    static func * (lhs: Confidence, rhs: Confidence) -> Confidence {
        return Confidence(lhs.value * rhs.value)
    }
}

extension Confidence: Comparable {
    static func < (lhs: Confidence, rhs: Confidence) -> Bool {
        return lhs.value < rhs.value
    }
}
```

Listing C.3: `Knowledge` module of CORTEX simulation, implemented in Swift.

```
struct Knowledge {
    let entities: Set<Entity>

    init(_ entities: [Entity] = []) {
        self.entities = Set(entities)
    }

    var problems: Set<Problem> {
        Set(entities.compactMap { $0 as? Problem })
    }

    var solutions: Set<Solution> {
        Set(entities.compactMap { $0 as? Solution })
    }

    var connections: Set<Connection> {
        Set(entities.compactMap { $0 as? Connection })
    }

    func getProblem(_ uuid: UUID) -> Problem? {
        problems.first(where: { $0.uuid == uuid })
    }

    func getSolution(_ uuid: UUID) -> Solution? {
        solutions.first(where: { $0.uuid == uuid })
    }

    func getConnection(_ uuid: UUID) -> Connection? {
        connections.first {$0.uuid == uuid}
    }

    func adding(_ newEntities: [Entity]) -> Knowledge {
        Knowledge(entities + newEntities)
    }

    func replacing(_ entity: Entity, with newEntity: Entity) ->
    Knowledge {
        let newEntities =
    entities.subtracting([entity]).union([newEntity])
        return Knowledge(Array(newEntities))
    }

    func connections(from problem: Problem) -> Set<Connection> {
        return connections.filter { $0.problemUUID == problem.uuid }
    }

    func connectedSolutions(for problem: Problem) ->
    Set<KnowledgeChain> {
        let findSolution: (Connection) -> KnowledgeChain? = {
    connection in
            guard let solution =
    getSolution(connection.solutionUUID) else {
                return nil
            }
            return KnowledgeChain(problem: problem, connection:
    connection, solution: solution)
        }
        return Set(connections(from:
    problem).compactMap(findSolution))
    }

    func problemFor(_ solution: Solution) -> Problem? {
        let connection = self.connections.first(where: {
    $0.solutionUUID == solution.uuid })
```

```
57          return self.problems.first(where: { $0.uuid ==
        connection?.problemUUID })
58      }
59  }
60
61  struct KnowledgeChain: Hashable {
62      let problem: Problem
63      let connection: Connection
64      let solution: Solution
65  }
```

Listing C.4: `Task` module of CORTEX simulation, implemented in Swift.

```
1  class Task {
2
3      let uuid: UUID = UUID()
4
5      typealias TaskResult = (knowledge: Knowledge, followUpTasks:
       Set<Task>)
6
7      func execute(on knowledgeBase: Knowledge, with params:
       SimulationParameters) -> TaskResult {
8          fatalError("Task#execute must be overloaded!")
9      }
10
11     func redundant(_ other: Task) -> Bool {
12         fatalError("Task#redundant must be overloaded!")
13     }
14 }
15
16 extension Task: Equatable {
17     static func == (lhs: Task, rhs: Task) -> Bool {
18         return lhs.uuid == rhs.uuid
19     }
20 }
21
22 extension Task: Hashable {
23     func hash(into hasher: inout Hasher) {
24         hasher.combine(uuid)
25     }
26 }
```

Listing C.5: `TaskPipeline` module of CORTEX simulation, implemented in Swift.

```
struct TaskPipeline {

    let waiting: Set<Task>
    let running: Set<Task>
    let finished: Set<Task>

    init(waiting: Set<Task> = Set(), running: Set<Task> = Set(),
    finished: Set<Task> = Set()) {
        self.waiting = Set(waiting)
        self.running = Set(running)
        self.finished = Set(finished)
    }

    var implementationTasks: Set<ImplementationTask> {
        let tasks = waiting.union(running).union(finished)
            .compactMap { $0 as? ImplementationTask }
        return Set(tasks)
    }

    var pendingResearchTasks: Set<ResearchTask> {
        let tasks = waiting.union(running)
            .compactMap { $0 as? ResearchTask }
        return Set(tasks)
    }

    func transition(incoming: Set<Task> = [], starting: Set<Task> =
    [], completed: Set<Task> = []) -> TaskPipeline {
        return TaskPipeline(
            waiting: waiting.subtracting(starting).union(incoming),
            running: running.subtracting(completed).union(starting),
            finished: finished.union(completed)
        )
    }

}
```

Listing C.6: `TaskProcessing` module of CORTEX simulation, implemented in Swift.

```
func completeAndApplyRunningTasks(pipeline: TaskPipeline,
    knowledge: Knowledge, parameters: SimulationParameters) ->
    (Knowledge, TaskPipeline) {
    let completedTasks = determineCompletedTasks(pipeline:
    pipeline, taskCompletionProbability:
    parameters.taskCompletionProbability)
    return applyCompletedTasks(completedTasks, pipeline: pipeline,
    knowledge: knowledge, params: parameters)
}

func determineCompletedTasks(pipeline: TaskPipeline,
    taskCompletionProbability: Int) -> Set<Task> {
    return pipeline.running.filter { _ in
        return Int.random(in: 0..<100) < taskCompletionProbability
    }
}

func applyCompletedTasks(_ tasks: Set<Task>, pipeline:
    TaskPipeline, knowledge: Knowledge, params:
    SimulationParameters) -> (Knowledge, TaskPipeline) {
    let totalResult: Task.TaskResult = tasks.reduce((knowledge,
    [])) { (memo: Task.TaskResult, task: Task) in
        let taskResult = task.execute(on: memo.knowledge, with:
    params)
        return (taskResult.knowledge,
    memo.followUpTasks.union(taskResult.followUpTasks))
    }
    let updatedPipeline = pipeline.transition(incoming:
    totalResult.followUpTasks, completed: tasks)
    return (totalResult.knowledge, updatedPipeline)
}

func startNextBatchOfTasks(pipeline: TaskPipeline, parameters:
    SimulationParameters) -> TaskPipeline {
    let freeWorkCapacity = max(parameters.workCapacity -
    pipeline.running.count, 0)
    let startingTasks =
    Set(pipeline.waiting.shuffled().prefix(freeWorkCapacity))
    return pipeline.transition(starting: startingTasks)
}
```

C.4 Validation of CORTEX Effects

C.4.1 Case Study

Examples for artifacts produced by practical application of CORTEX in large-scale organization.

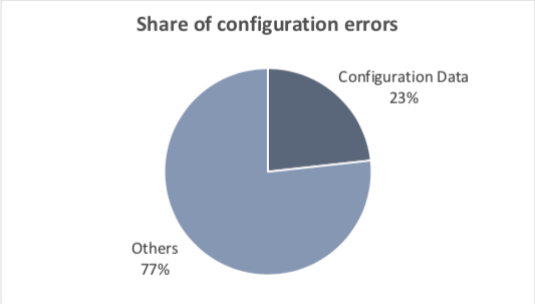


Figure C.4: Example for analysis of the cause of errors, isolating a particular problem area [123].

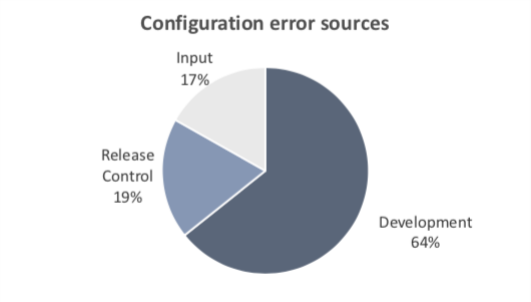


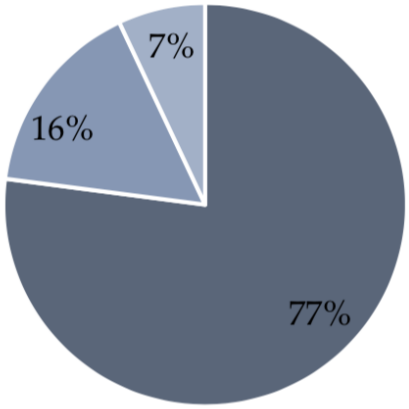
Figure C.5: Example for analysis of error Share of configuration data errors within different departments [123].

Problem Title	Problem Description
Comprehension	Cognitively grasp the meaning behind configuration knowledge.
Transmission	Transporting configuration knowledge between departments or stakeholders.
Creation	Creating or editing configuration knowledge.
Verification	Verifying correctness, consistency and completeness of configuration knowledge.

Table C.3: Example for top-level problems discovered through analysis of the problem domain [123].

Category	Reach	Impact	Confidence	Complexity	Total
Comprehension	800	3	100%	2	1200
Transmission	800	2	90%	2	720
Creation	800	2	90%	4	360
Verification	800	3	100%	8	300

Table C.4: Example for domain problems prioritized using the RICC model [123].



Participants:	414
Target group:	1281
Participation:	32 %

Constructors	319
Executives	66
No role submitted	29

Figure C.3: Example for participants and roles of stakeholder survey for problem discovery [123].

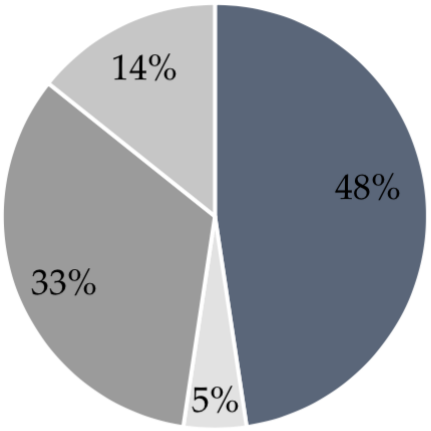
Category	**Solution**	**Kano**	**Effort**
Comprehension	Tree visualization	Basic	High
Comprehension	Block visualization	Excitement	Low
Comprehension	Matrix visualization	Basic	Moderate
Comprehension	Identifier key translation	Excitement	Low
Creation	Matrix optimization	Performance	Moderate
Creation	Variance matrix editor	Basic	Moderate
Creation	Block editor (drag&drop)	Excitement	High
Creation/Transmission	Expression generation (matrix)	Performance	Low
Creation/Transmission	Expression generation (graphic)	Excitement	High
Creation	Expression formatting	Excitement	High
Comprehension	Bracket coloring	Basic	Low
Verification	Syntax verification	Performance	Moderate
Creation	Search engine	Excitement	High

Table C.5: Example for evaluation of solution variants (i.e., connections) and estimation of corresponding cost [123].

Category	Name
Comprehension	Bracket coloring
Comprehension	Block visualization
Comprehension	Matrix visualization
Comprehension	Identifier key translation
Creation	Matrix optimization
Creation	Variance matrix editor
Creation	Expression generation (matrix)
Creation	Expression formatting
Verification	Syntax verification

Table C.6: Example for a feature backlog representing decisions on specific solution variants to specific problems [123].

C.4.2 User Survey



Participants:	21
Target group:	100
Participation:	21 %

Release control experts:	10
Constructors:	7
Others:	3
Release control implementers:	1

Figure C.6: Example for participation and roles of users within the feedback survey [123].

Category	Question
Design/GUI	How do you evaluate the tool regarding clarity?
Design/GUI	How do you evaluate the tool regarding the graphical user interface?
Design/GUI	How do you evaluate order and boundaries of presented information?
Design/GUI	Proposed improvements regarding design/GUI.
Usability	How do you evaluate the tool regarding intuitiveness?
Usability	How do you evaluate the tool regarding performance?
Usability	How do you evaluate user comfort?
Usability	Proposed improvements regarding usability.
Functionality	How do you evaluate complexity reduction through visualization?
Functionality	How satisfied are you with the generated configuration knowledge?
Functionality	How do you evaluate the syntax checking mechanism?
Functionality	How do you evaluate the tool regarding robustness?
Functionality	Proposed improvements regarding functionality
Functionality	Desired features for additional releases
	The tool ...
Innovation	... improves quality of configuration knowledge?
Innovation	... saves time when creating or editing configuration knowledge?
Innovation	... improves comprehensibility of configuration knowledge?
Innovation	... enables a structured and methodic derivation of configuration knowledge?
Innovation	... lowers overall complexity of configuration knowledge for users?
Innovation	... offers innovative functionality not provided by other applications?
Overall	Which overall rating would you grant the tool?
Overall	Would you recommend the tool to your colleagues?

Table C.7: Example of categories and questions within the stakeholder feedback survey [123].

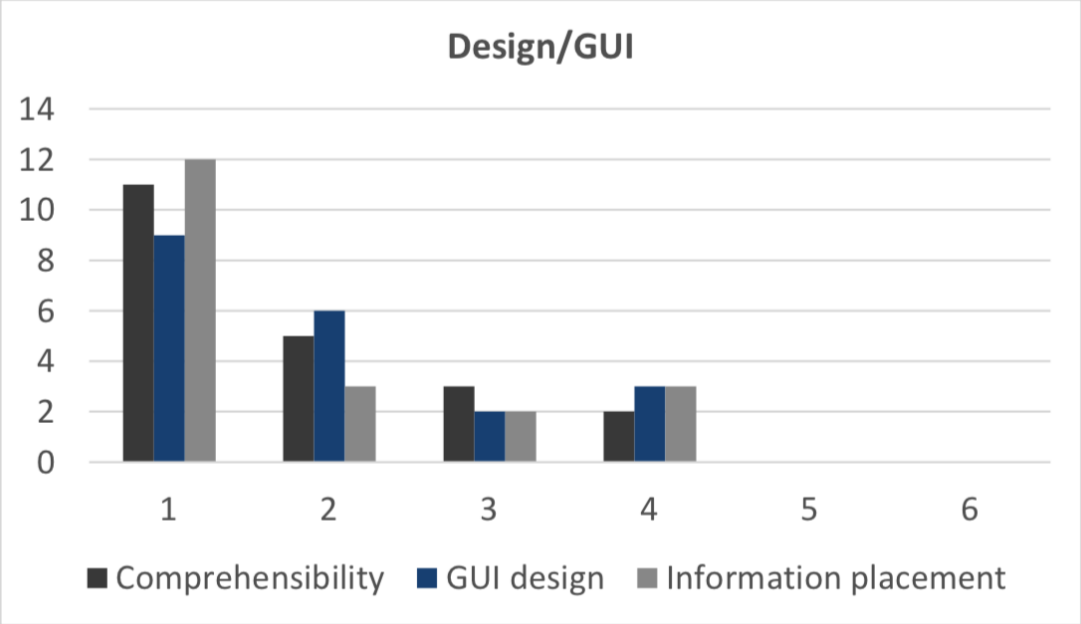


Figure C.7: Example for detailed design grades [123].

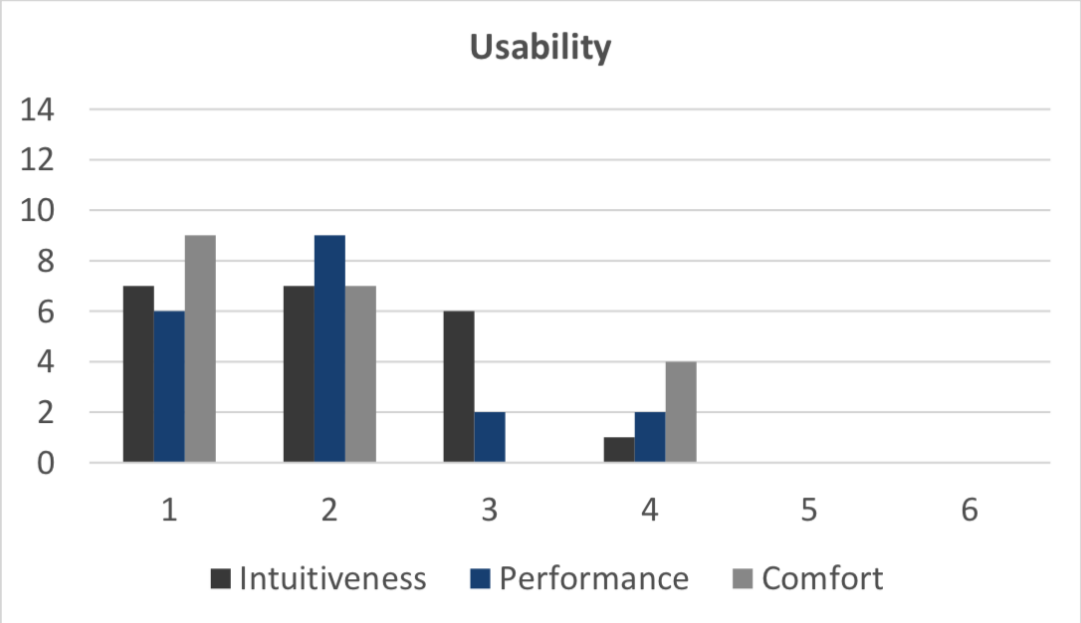


Figure C.8: Example for detailed usability grades [123].

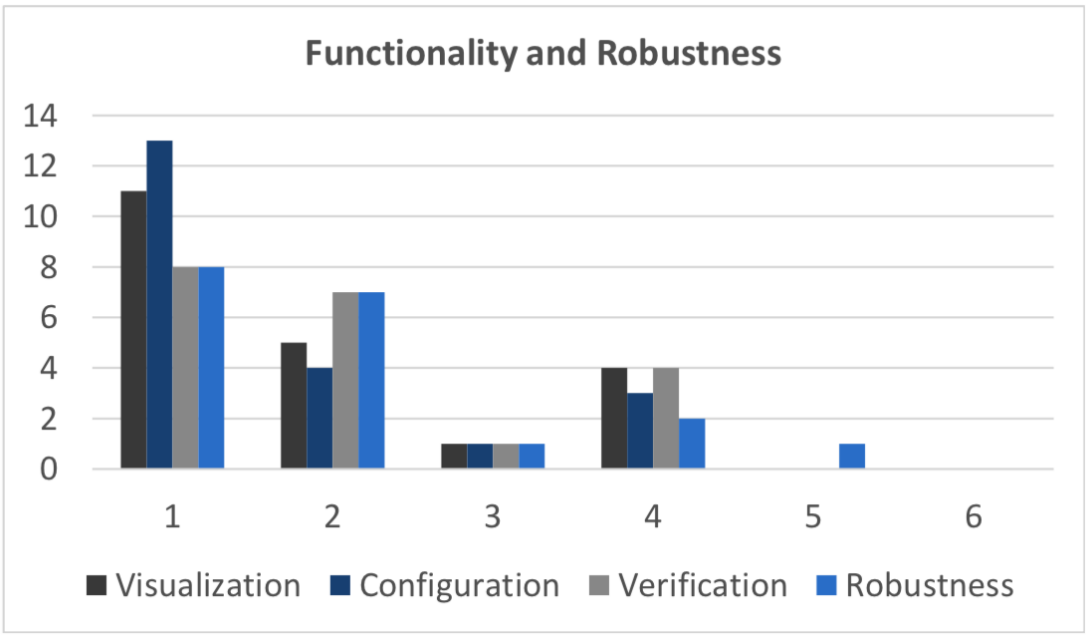


Figure C.9: Example for detailed functionality/robustness grades [123].

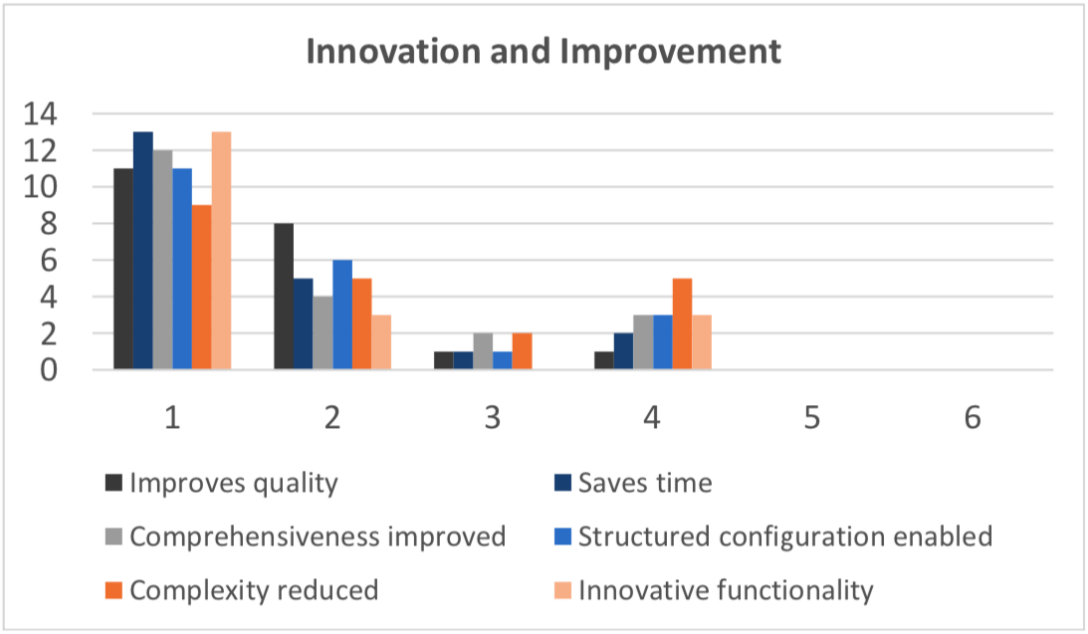


Figure C.10: Example for detailed innovation/improvement grades [123].

Feedback	Efficiency	Effectiveness	Quality
The tool is perfect when verification mechanisms using official databases are available.	mildly positive	mildly positive	positive
The visualization functionality is really well done and useful, but other configuration functionality would be favorable.	neutral	mildly positive	mildly positive
Finally a well implemented standard for resolving the current communication issues between release control and construction.	positive	positive	positive
The novel visualization mechanisms really help for understanding configuration knowledge.	neutral	positive	positive
The configuration generation works very well and makes configuring new parts easy.	neutral	positive	positive
The translation from identifier key to description will be really useful in daily work.	neutral	positive	positive
It is amazing that you have implemented all this with adequate robustness in such a short amount of time.	positive	positive	mildly positive
I would like to have more extensive mechanisms for generating configurations, currently limited to one configuration at a time.	neutral	mildly negative	neutral
If you included a search function and make the matrix easier to work with, I would have really liked the tool.	mildly negative	mildly negative	mildly negative
Without connection to databases, it is unfortunately not as valuable and does only help in terms of visualization.	mildly negative	negative	neutral

Table C.8: Example for feedback received during release events and estimations regarding impact on relevant metrics [123].

Appendix D

RADAR

D.1 Suitability of Existing Tools

Details on tool evaluation, data and reports from Widera [139].

Category	Selected Tool	FR Score	NFR Score
Task Mgmt.	Atlassian Jira	88/88	19/24
Spreadsheet	Microsoft Excel	49/88	13/24
Wiki	Atlassian Confluence	43/88	13/24
Project Mgmt.	OmniPlan	34/88	3/24
Requirement Mgmt.	Siemens Polarion	83/88	17/24
Mind Mapping	FreeMind	54/88	12/24
Modeling	Draw.io	56/88	12/24

Table D.1: Rating of representative tools per category, based on functional and nonfunctional requirements.

Analysis of Requirement Management Software According to the State of Agile Report 2018, 46% of all projects use requirement management software[1]. When the problem domains become complex and the requirements change frequently, it is hard for a human to capture them [140]. During the course of a project, requirements become more concrete, more complex and more detailed [142]. Requirement management tools help to store these requirements, enable relationships between them and assist in other development activities [142]. Since the CORTEX workflow handles requirements in the form of problems, this category was selected for modeling it.

Taking the requirements for requirements management systems [142] into account, we chose the Polarion Software owned by Siemens as a well suited representative for this category. Because the focus of this software is managing requirements, there is no feature to model the difference between solutions, uncertainties and problems. We can only create two different types of entities: requirements and tasks. To still distinguish between problems, uncertainties and solutions, we can create an own category for each type. These categories are saved to the requirement entity and we can filter and query for them. Requirements and tasks can connect to multiple requirements. This way the n to m relations between problems and subproblems can be modeled. Furthermore, requirements can store information and values. These values can be used for ranking them inside the user interface. We can model work items as tasks and connect them to requirements in a n:m relationship. They can represent research or implementation tasks and get distinguished through their category. Tasks can be prioritized and put in a queue. The user interface of Polarion is kept simple, which makes it easy to get used to. A big advantage of this software is that is can show the relationship between entities in a tree structure, which is described in the requirements.

[1] VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

Analysis of Wikis For information management wikis are a well-established way to collect and store knowledge. They are used in 62% of agile projects[2] and therefore worth looking into. We chose Atlassian Confluence for this category as a well-known wiki software for companies [160].

Since Wikis primary focus is information management, they are a great tool to store information belonging to a problem or solution. Diagrams, files and tables are one of many options to capture this information. But because of this focus it is difficult to model different entities with this software. We can use the page hierarchy to model the division from problem into subproblems, but only in a one to n relation. Duplicate problems have to be linked via hyperlinks. If solution space and problem space are kept separate in the hierarchy, we have to use hyperlinks to capture the links between problems and the belonging solutions. Therefore, it is difficult to keep an overview over which problems already have a solution and which do not. Since wikis are designed for information management, there is no easy way to model work items. Pages can be used for each item, but they cannot be put in a queue or sorted for different values.

Analysis of Spreadsheet Tools The *12th Annual State of Agile Report* by VersionOne Inc. [13] states that 65% of all projects use Spreadsheets as a project management tool[3]. This number steadily decreased from 74% in 2015[4], but it is still significant enough to take into account. According to the agile report 46% of projects use Excel as an agile management tool, but only 31% of them would recommend it to others. This shows that excel is in high usage, but the users are not satisfied with it. However, Excel is the oldest and most popular tool in this category with a market share of 60% [??], so we chose it as the representative tool.

Since a spreadsheet tool saves all data in rows and columns, all entities and information needed to be stored this way. To model the different kinds of entities, we can use different sheets. We can save problems and subproblems in different sheets as well. This way a subproblem can link to multiple problems with the use of hyperlinks and vice versa. Solutions can be linked the same way. Columns can be used so save additional information and values of entities . If the value is numeric, they can be sorted according to it. Another possibility is to use background colors to visualize the certainty about a value or information. Work item also have a separate sheet and the type is modeled in a column. The queue can get realized with an index column in the work item sheet.

This setup is not convenient to operate with, since we cannot visualize relation- ships between entities and cannot view entities separately. It is not convenient to model the relationships via hyperlinks and the user interface is not designed for such tasks. Spreadsheets are convenient to store multidimensional data, but not to model hierarchical problems.

[2] VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`.

[3] VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`.

[4] VersionOne Inc. *9th Annual State of Agile Report*. 2015. URL: `https://explore.versionone.com/state-of-agile/9th-annual-state-of-agile-report-2`.

Analysis of Mind-map Tools Mind maps are not listed under the most common management tools, but they still have a right to be in the evaluation. Since problems, subproblems, solutions and work items are highly connected, we added mind-mapping tools to the evaluation. Moreover, it can yield advantages to visualize the links between these entities. We chose the software FreeMind as a representative tool, since it is a well-known open source software with around 3000 downloads every day.[5]

A big problem of using a mind map tool to model the workflow is that in mind maps nodes usually only have one predecessor. As a result, we can not model many-to-many relationships. Problems can have multiple subproblems, but if a subproblem is part of more than one problem, this relationship cannot be displayed. A workaround for this problem is to store duplicate nodes and link between them. The same problem exists for solutions. Information and values can be added to each entity but ranking them according to this information is not possible. Distinguishing problems and solutions is only possible through different icons. Work items can also be connected to solutions or problems, but they cannot be put in a queue.

Overall, mind maps are a great tool for visualizing the connectivity between entities. Entities can be created fast, but it is not convenient to operate with these entities.

Analysis of Modeling Tools The modeling tool category was taken into account, because it is the freest way of modeling graphical relationships without any restrictions as seen in mind mapping tools. Choosing a special tool for it was not of big importance, because modeling tools usually provide the same functionality with different user interfaces. We chose the web application Draw.io as the representative for the modeling tool category. Modeling tools can easily be compared to mind map tools, but an advantage of drawing tools is that it is possible to model many-to-one relationships. Furthermore, problems, solutions and work items can be distinguished easier, because different shapes can be used for the different entities. Uncertainty can be displayed with dashed lines around an entity or a dashed connection between two entities.

Figure D.1 shows an example graph of modeled entities and their connections. On the downside, it is not practical to save information about the entities. This would make it more difficult to keep an overview over the modeled space and see the links between the entities. Adding values is possible, but we cannot ranked them according to it. Modeling tools have advantages and disadvantages to mind maps, but they are a good way to display the connectivity of the problems and solutions. This model design might make a good extension for other tools.

Analysis of Project Management Tools Traditional project management tools focus on capturing different tasks, which need to be completed and distributing them among a big team. 40% of the respondents in the Agile State Report mentioned, that project management tools were used in their projects. Microsoft project is one of the biggest examples, but Omni plan is an alternative for Mac, which we used in this evaluation.

[5] SourceForge. *FreeMind Download Statistics*. 2019. URL: `https://sourceforge.net/projects/freemind/files/stats/timeline`.

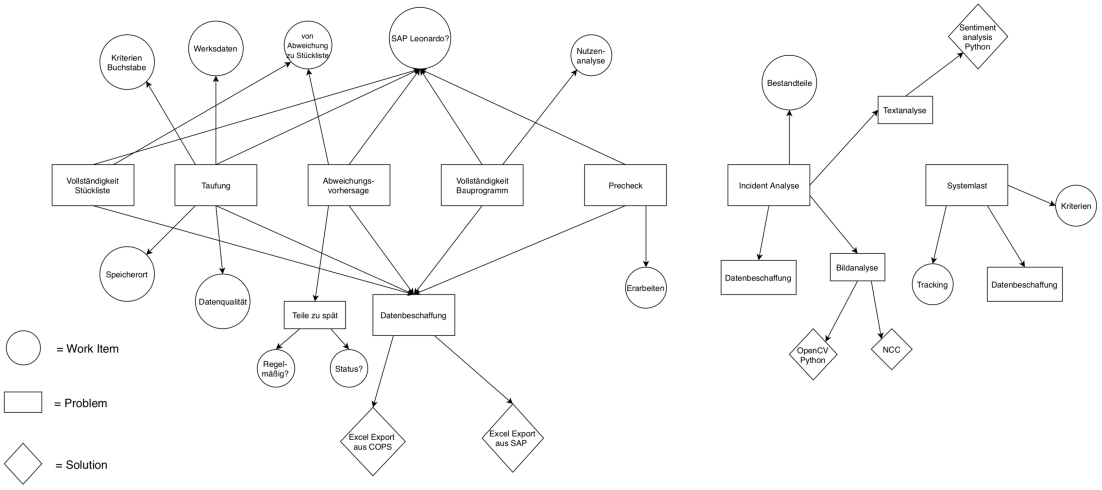


Figure D.1: Graph of entities created with Draw.io, the representative for the modeling tool category.

For modeling the CORTEX workflow with its different entities, this tool was not suited. The task of distributing and prioritizing work items, can be done well, but problems or solutions can only be captured as tasks. Tasks can only be connected to other tasks in 1:n relationships. It is not possible to differentiate problems from solutions and work items. Furthermore, it is not possible to save additional information or values to the tasks.

Analysis of Task Boards Task boards are a tool category with consistently high usage in agile project management according to VersionOne Inc. [13]: In 2015 they were used from 82% and slightly decreased to a usage of 71% in 2017[6]. The usage of kanban boards on the other hand, increased from 63% to 74%. It can be assumed that the decrease of task boards leads back to the increase of kanban boards. For both these categories the same tool can be used. An overall number of 58% of the respondents are using Atlassian JIRA.

[6] VersionOne Inc. *12th Annual State of Agile Report*. 2018. URL: `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`.

D.2 Validation of RADAR Effects

Details on reference implementation and validation case study from Widera [139].

D.2.1 Use of Jira Functionality

As a first step, we created the four new issue types and included them in an issue type scheme for the project. These four issue types are problem, solution, uncertainty and work item. Each issue type has different attributes as described in the system design (section 4.2).

When creating a new issue, first the issue type has to be determined (see fig. D.2). After creating the issue, the user has to capture its attributes. The attributes depend on the selected issue type. We created a custom field for each attribute needed and made it available for only this issue type through the creation of multiple screen schemes. These screen schemes display the most important attributes of the issue, depending on the issue type. As an example, fig. D.3 depicts the problem dialog which appears, when a user edits a problem issue. This dialog displays the problem screen scheme, which contains all relevant problem information for the workflow including the option to create links to different issues.

We modeled all possible links between issues in the issue linking scheme (see fig. D.4). All links have an outward description and an inward description. The outward description describes the link from the origin's perspective. For example, when the user links a solution to a problem, the link in the problem description is "is solved by", because the problem is solved by the solution. In the description of the solution the link is called "solves", because the solution solves the problem. These links are based on the links derived in section 4.2.6.

Figure D.5 shows an example of the issue link section of a problem issue. Here Jira displays all incoming and outgoing links of the selected issue. Besides the link description and the linked issue name, the status of the issue is shown. This example problem depicted in fig. D.5 has different incoming and outgoing links. It is a subproblem of the problem `MAWIDERA-1` and the link is displayed with the according link description "parts into". Furthermore, the problem "is solved by" a solution which is yet unimplemented and has an uncertainty issue linked to it which is already resolved.

After Jira saves an issue, the user can select it and see the details of the issue. These details are parted into different sections. One section described earlier is the section about the issue links. The details section about an issue (see fig. D.6) is configured through a screen scheme. Our screen schemes are configured to only display the relevant attributes for each issue.

For the search of issues, Jira offers a wide variety of possibilities. The user can filter for the issue types we created and all configured attributes. Figure D.7 shows the search for the problem issue type.

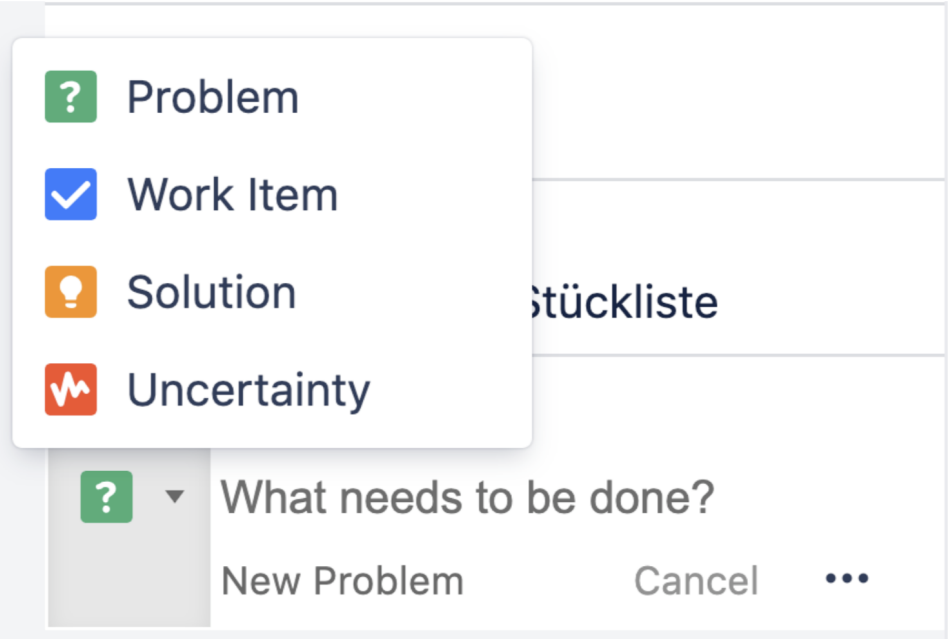


Figure D.2: Creation of a new entity in Jira. Four different issue types can be selected.

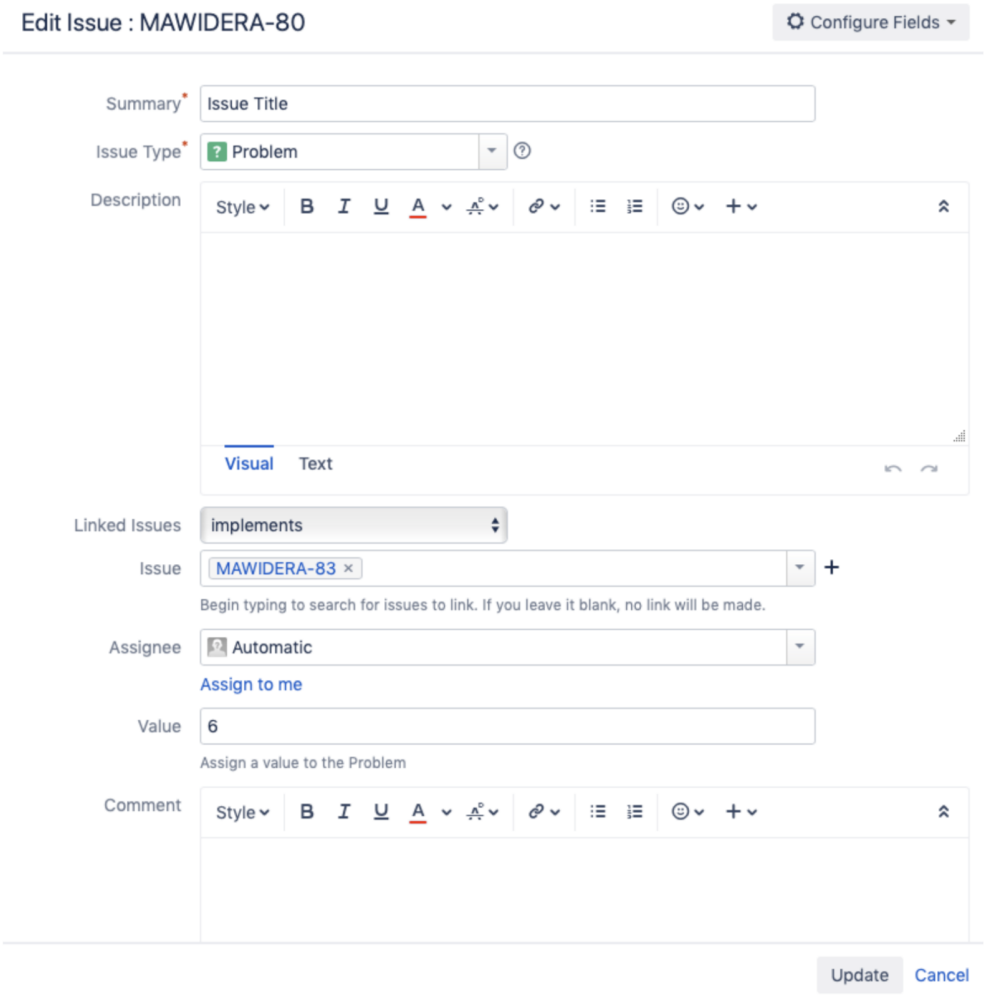


Figure D.3: Dialog displayed when a problem is edited. All important information for the workflow can be captured.

Name	Outward Description	Inward Description
Implementation	is implemented by	implements
Research	is clarified by	clarifies
Solution	is solved by	solves
Subproblem	parts into	is part of
Uncertainty	requires clarification of	complicates

Figure D.4: Table of the modeled issue links in Jira. It depicts the different link names, outward and inward descriptions.

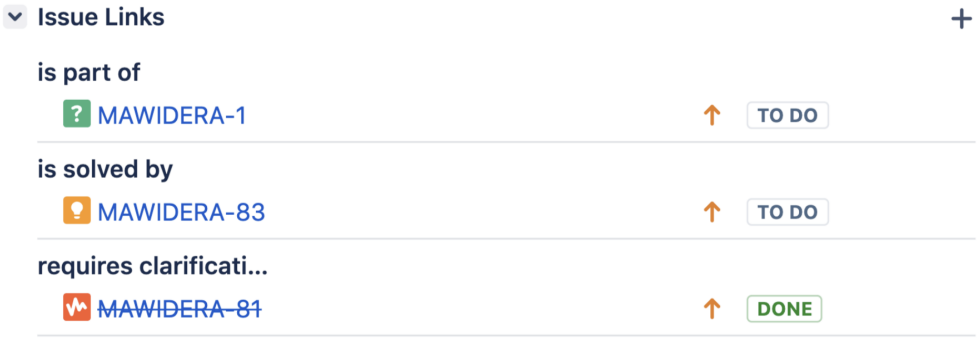


Figure D.5: The issue link section of a problem issue. It depicts the incoming and outgoing links of the problem issue.

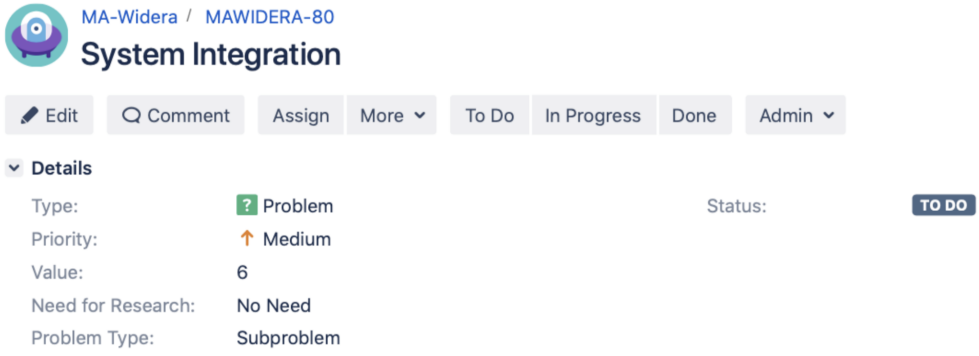


Figure D.6: Details section of a problem issue. It displays the relevant information for the workflow.

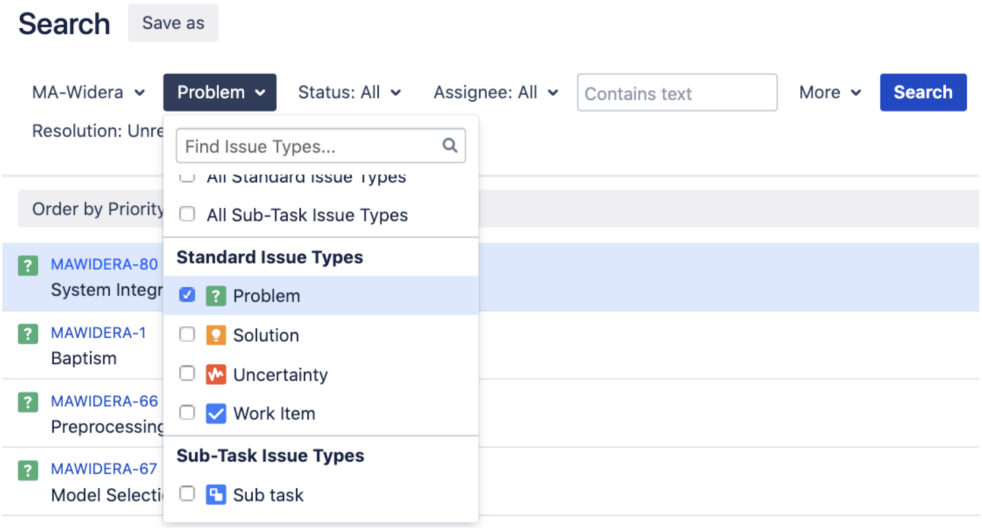


Figure D.7: Jira Issue search for issues of the type problem.

D.2.2 Extension of Jira Functionality

The last section demonstrated the options Jira offers to fit to the object design. But since Jira provides the functionality to install third party add-ons, we extended out reference implementation through different add-ons, which suit to the workflow and fulfill the functional requirements. The following two sections introduce two different add-ons which extend our Jira implementation.

Issue Links Viewer Add-on Through issue links, Jira offers the functionality to save connections between issues. These links are displayed in the issue link section. When the user wants to see for instance the parent of an issue, he has to go to the issue link section and click on the parent issue. An overview over the issues in a graphical way is not possible. This overview is included in our functional requirements and highly helpful for a better understanding of the existing entities.

We searched for already existing add-ons, since the implementation of a Jira add-on is expensive. Issue Links Viewer is an add-on which provides exactly the functionality needed to visualize the links between issues. When an issue is displayed, a the add-on created a section which displays the issue link graph. Figure D.8 shows a small example issue link graph created by the Issue Links Viewer. This graph displays all parents and children of the issue. In the example graph the problem is a subproblem of the problem `MAWIDERA-1` and has a solution and uncertainty linked to it. The links are labeled with the according issue link labels. This graph visualized the links displayed in fig. D.5.

The user can expand each displayed issue by double clicking on it. When an issue has many links, this issue link graph can become large. Figure D.9 shows the link graph of a root problem in our case study. The problem is parted into four subproblems and has an uncertainty connected. These subproblems have solutions or other subproblems connected to them. On the last level of the tree, the work items are displayed consisting of research and implementation tasks.

Through the Jira issue search depicted in fig. D.7 the user is able to filter for issue types and issue attributes. After the result list returns, Issue Links Viewer offers the functionality to display the received results in an individual graph. This way only entities of a special issue type can be visualized. For example, fig. D.10 depicts a part of the visualized graph for an example query. The query filtered all issues for the issue type problem. We can use this graph to get an overview over all encountered problems and their relations.

Scriptrunner Add-on To better assist in decision support, we can make use of the values of problems, cost of solutions and degrees of uncertainties. As explained in the process of finding the next work item (see section 3.2.3.2), the process of indicating where work is needed can be automated when all issues have a cost, value or degree saved to it.

We could implement this tree traversal algorithm through a new self developed add-on, but we found an existing add-on to realize this functionality. The add-on ScriptRunner for Jira provides a toolset to the user for automating and enhancing Jira through the use

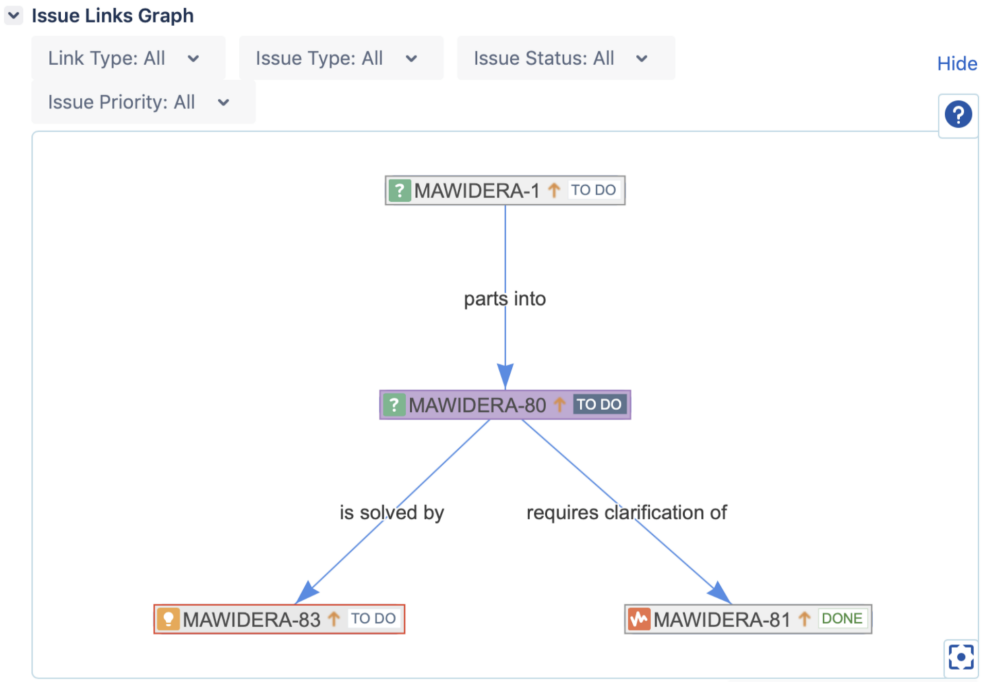


Figure D.8: A graph, visualizing the issue links of a single issue. It is generated by the Jira add-on Issue Links Viewer.

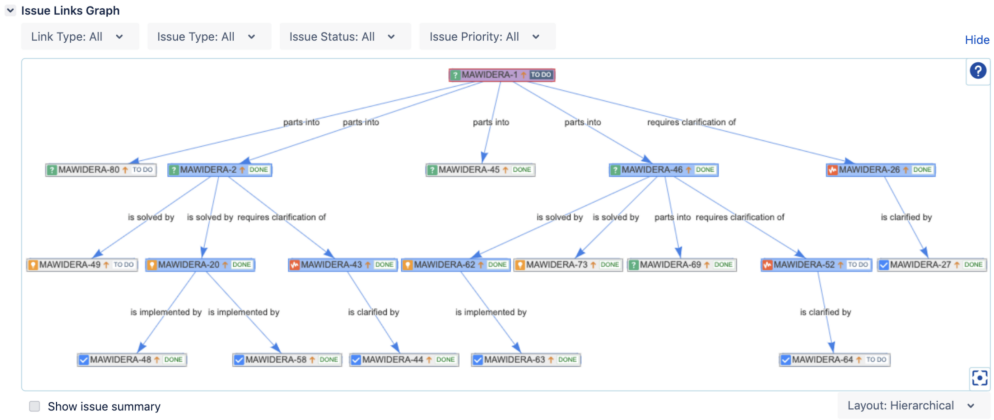


Figure D.9: A issue link graph, depicting all children of a problem issue from the case study. It is generated by the Jira add-on Issue Links Viewer

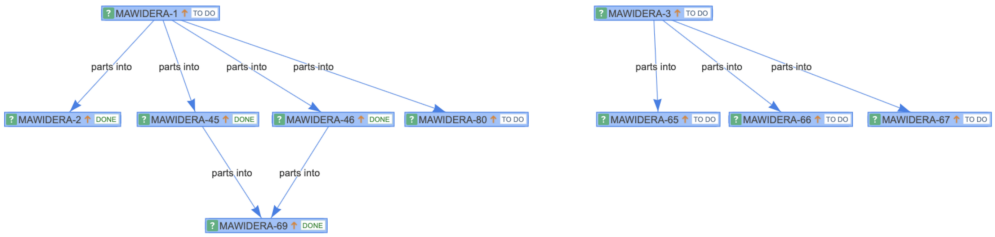


Figure D.10: Result of the Jira issue search visualized from the add-on Issue Links Viewer. It depicts only problem issues and their connections.

of Groovy scripts[7]. We can implement the algorithm described in section 3.2.3.2 with a groovy script to suggest solutions with a high value/cost ratio or uncertainties with a high degree which are connected to a high value problem.

Besides finding issues to work on, we can further use Scriptrunner for scripted fields. As explained in section 4.2.6, a problem needs different attributes saved to it. The attribute about the need for research and about the root problem have to be updated every time new uncertainties or problems are connected to it. This is a troublesome activity which could be automatized easily. Scriptrunner enables us to transform these issue fields into scripted fields which calculated the values automatically.

In our implementation, we changed the the issue fields about need of research and the problem origin to scripted fields. The need for research field determines how many unresolved uncertainties are connected to the problem and based on that assigns a value to it. The issue field about the root problem checks if the problem has a problem connected to it as its parent and determines the corresponding value. Listing D.1 shows the script used to determine the problem type.

Listing D.1: The groovy script to compute the problem type custom field with the Jira add-on Scriptrunner

```
1  import com.atlassian.jira.issue.Issue
2  import com.atlassian.jira.issue.link.IssueLink;
3  import com.atlassian.jira.component.ComponentAccessor;
4  def predecessor = false;
5  List<IssueLink> allInwardIssueLink =
       ComponentAccessor.getIssueLinkManager()
6
       .getInwardLinks(issue.getId());
7  for (Iterator<IssueLink> inIterator =
       allInwardIssueLink.iterator(); inIterator.hasNext();) {
8          IssueLink issueLink = (IssueLink) inIterator.next();
9          def linkedIssue = issueLink.getDestinationObject()
10         String type = linkedIssue.getIssueType().getName();
```

[7]ScriptRunner for Jira. URL: https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira

```
11          if(type.contains("Problem")) {
12                  predecessor = true;
13  } }
14  return (predecessor) ? "Subproblem" : "Root Problem"
```

List of Figures

List of Tables

Bibliography

[1] B. Fitzgerald and K. J. Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189.

[2] W. W. Royce. "Managing the development of large software systems: concepts and techniques". In: *ICSE '87 Proceedings of the 9th international conference on Software Engineering* (1987).

[3] M. Broy and A. Rausch. "Das neue V-Modell® XT". In: *Informatik-Spektrum* 28.3 (2005), pp. 220–229.

[4] P. Kruchten. *The Rational Unified Process: An Introduction*. 2nd. Addison-Wesley, 2000.

[5] K. Beck. "Embracing change with extreme programming". In: *Computer* 32.10 (1999), pp. 70–77.

[6] K. Schwaber. "SCRUM Development Process". In: *Business Object Design and Implementation*. Springer London, 1997, pp. 117–134.

[7] M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit*. Addilson-Wesley, 2003.

[8] J. Humble and D. Farley. *Continuous Delivery*. 1st ed. Addison-Wesley Professional, 2010.

[9] B. Fitzgerald and K.-J. Stol. "Continuous software engineering and beyond: trends and challenges". In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering - RCoSE 2014* (2014), pp. 1–9.

[10] J. Bosch. *Continuous Software Engineering*. Springer, 2014.

[14] L. Northrop et al. *Ultra-Large-Scale Systems - The Software Challenge of the Future*. Carnegie Mellon University, Jan. 2006.

[15] E. Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[16] J. Highsmith and A. Cockburn. "Agile software development: the business of innovation". In: *Computer* 34.9 (2001), pp. 120–127.

[17] D. Zowghi and V. Gervasi. "The Three Cs of Requirements: Consistency, Completeness, and Correctness". In: *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality*. 2002, pp. 155–164.

[18] C. Denger and T. Olsson. "Quality assurance in requirements engineering". In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Springer Berlin Heidelberg, 2005, pp. 163–185.

[20] M. Coram and S. Bohner. "The impact of agile methods on software project management". In: *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS'05*. IEEE, 2005, pp. 363–370.

[21] K. Nageswara Rao et al. "A Study of the Agile Software Development Methods, Applicability and Implications in Industry". In: *International Journal of Software Engineering and its Applications* 5.2 (2011).

[22] S. Krusche et al. "Chaordic Learning: A Case Study". In: *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. ICSE-SEET '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 87–96.

[23] T. Dybå and T. Dingsøyr. "Empirical Studies of Agile Software Development: A Systematic Review". In: *Information and Software Technology* 50.9–10 (Aug. 2008), pp. 833–859.

[24] Prowareness. *Ist das Scaled Agile Framework (SAFe) wirklich agil?* 2015. URL: `https://www.scrum.de/ist-das-scaled-agile-framework-safe-wirklich-agil/`.

[25] J. Bosch. "Building products as innovation experiment systems". In: *Lecture Notes in Business Information Processing*. Vol. 114 LNBIP. 2012, pp. 27–39.

[26] A. Sillitti and G. Succi. "Requirements Engineering for Agile Methods". In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Springer Berlin Heidelberg, 2005, pp. 309–326.

[27] H. H. Olsson and J. Bosch. "The HYPEX Model: From Opinions to Data-Driven Software Development". In: *Continuous Software Engineering*. Springer, 2014, pp. 155–164.

[28] M. Ekström and Í. D. Þorvaldsson. "Predicting and Analyzing Feature Value when R&D is an Experiment System". PhD thesis. University of Gothenburg, 2016.

[29] F. Fagerholm et al. "The RIGHT model for Continuous Experimentation". In: *Journal of Systems and Software* 123 (2017), pp. 292–305.

[30] O. Rissanen and J. Münch. "Continuous Experimentation in the B2B Domain: A Case Study". In: *Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015*. 2015, pp. 12–18.

[31] S. G. Yaman et al. "Introducing continuous experimentation in large software-intensive product and service organisations". In: *Journal of Systems and Software* (July 2017).

[32] T. Eisenmann et al. "Hypothesis-Driven Entrepreneurship: The Lean Startup". In: *Harvard Business School Background Note 812-095* 44 (2011), pp. 1–23.

[33] J. Bosch et al. "The early stage software startup development model: A framework for operationalizing lean principles in software startups". In: *Lecture Notes in Business Information Processing*. Vol. 167. 2013, pp. 1–15.

[34] A. Fabijan et al. “Early value argumentation and prediction: An iterative approach to quantifying feature value”. In: *Lecture Notes in Computer Science*. Vol. 9459. 2015, pp. 16–23.

[35] H. H. Olsson and J. Bosch. “Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation”. In: *Lecture Notes in Business Information Processing*. Vol. 210. 2015, pp. 154–166.

[36] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. 3rd ed. ACM SIGSOFT Software Engineering Notes, 2000.

[37] A. Aurum and C. Wohlin. *Engineering and Managing Software Requirements*. Springer, 2005.

[38] D. H. Jonassen. “Toward a design theory of problem solving”. In: *Educational Technology Research and Development* 48.4 (2000), pp. 63–85.

[39] J. Funke. “Solving complex problems: Exploration and control of complex systems”. In: *Complex problem solving: Principles and mechanisms*. Ed. by R. J. Sternberg and P. A. Frensch. Taylor and Francis, 2014. Chap. 6.

[40] D. Beuche and M. Dalgarno. “Software product line engineering with feature models”. In: *Overload Journal* 78 (2007), pp. 5–8. URL: `https://www.accu.org/var/uploads/journals/Overload78.pdf%7B%5C#%7Dpage=6`.

[41] A. Ngo-The and G. Ruhe. “Decision support in requirements engineering”. In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Springer Berlin Heidelberg, 2005, pp. 267–286.

[42] F. Knight. “From Risk, Uncertainty, and Profit”. In: *The economic nature of the firm*. Ed. by R. S. Kroszner and L. Putterman. Cambridge University Press, 2009, pp. 52–57.

[43] D. Hubbard. *The Failure of Risk Management*. John Wiley & Sons, 2012.

[44] C. Chapman and S. Ward. *Project Risk Management: Processes, Techniques and Insights*. 2nd. John Wiley & Sons, 2003.

[45] R. Atkinson. “Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria”. In: *International Journal of Project Management* 17.6 (1999), pp. 337–342.

[56] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus, 2011.

[57] H. W. Rittel and M. M. Webber. “Wicked problems”. In: *Man-made Futures* 26.1 (1974), pp. 272–280.

[58] B. Roy and D. Bouyssou. *Aide multicritère à la décision: méthodes et cas*. Economica Paris, 1993.

[59] H. A. Simon. *The New Science of Management Decision (rev. ed.)* Englewood Cliffs, NJ: Prentice-Hall, 1977.

[60] D. Noble and H. W. J. Rittel. "Issue-Based Information Systems for Design". In: *Computing in Design Education [ACADIA Conference Proceedings]*. CumInCAD, Oct. 1988, pp. 275–286.

[61] S. Klepper et al. "Continuous Innovation and Experimentation in Complex Problem Domains: Problem Solving and Decision Support as a Starting Point for a Unified Process Framework". In: *Software Engineering und Software Management*. Gesellschaft für Informatik, 2018, pp. 191–194.

[62] K. Dalkir. *Knowledge Management in Theory and Practice*. Routledge, 2013.

[63] R. J. Wieringa. *Design science methodology: For information systems and software engineering*. 2014, pp. 1–332. ISBN: 9783662438398.

[64] A. J. Onwuegbuzie and R. Frels. "Seven Steps to a Comprehensive Literature Review". In: 23.2 (2016), pp. 48–64.

[65] H. Snyder. "Literature review as a research methodology: An overview and guidelines". In: *Journal of Business Research* 104.August (2019), pp. 333–339.

[66] H. W. Rittel and M. M. Webber. "Dilemmas in a general theory of planning". In: *Policy Sciences* 4.2 (1973), pp. 155–169.

[67] J. Conklin. *Dialogue Mapping: Building Shared Understanding of Wicked Problems*. John Wiley & Sons, 2006.

[68] F. P. Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.

[69] L. Suchman. *Human-machine reconfigurations: Plans and situated actions*. Cambridge University Press, 2007.

[70] K.-J. Stol and B. Fitzgerald. "A Holistic Overview of Software Engineering Research Strategies". In: *2015 IEEE/ACM 3rd International Workshop on Conducting Empirical Studies in Industry*. IEEE Press, 2015, pp. 47–54.

[71] S. B. Hulley et al. *Designing Clinical Research*. Lippincott Williams & Wilkins, 2007.

[72] A. Fabijan et al. "The Evolution of Continuous Experimentation in Software Product Development". In: *Proceedings of the 39th International Conference on Software Engineering ICSE'17*. 2017.

[73] I. Jacobson et al. *The Unified Software Development Process*. Addison-Wesley, 1999.

[74] S. Krusche and B. Bruegge. "CSEPM: A Continuous Software Engineering Process Metamodel". In: *Proceedings of the 3rd International Workshop on Rapid Continuous Software Engineering*. RCoSE '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 2–8.

[75] P. Berander and A. Andrews. "Requirements Prioritization". In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Springer Berlin Heidelberg, 2005, pp. 69–94.

[76] S. Trieflinger et al. “How to Prioritize Your Product Roadmap When Everything Feels Important : A Grey Literature Review”. In: *International Conference on Engineering, Technology and Innovation (ICE)*. June. 2021.

[77] J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[78] L. Börries. *Computerunterstützung der Argumentation in Gruppen*. Ed. by H. Krcmar. Wiesbaden: Deutscher Universitätsverlag, 1997.

[79] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994.

[80] T.-M. Hesse et al. “Experiences with Supporting the Distributed Responsibility for Requirements through Decision Documentation”. In: *Softwaretechnik-Trends* 35.1 (2015).

[81] W. J. Brown et al. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Ed. by T. Hudson. John Wiley & Sons, 1998.

[82] G. Visaggio. “Value-based decision model for renewal processes in software maintenance”. In: *Annals of Software Engineering* 9.1-2 (2000), pp. 215–233.

[83] G. Ruhe et al. “Quantitative WinWin: a new method for decision support in requirements negotiation”. In: *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, pp. 159–166.

[84] G. Ruhe. “Software Engineering Decision Support – A New Paradigm for Learning Software Organizations”. In: *Advances in Learning Software Organizations*. Ed. by S. Henninger and F. Maurer. Springer Berlin Heidelberg, 2003, pp. 104–113.

[85] W. E. Walker et al. “Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support”. In: *Integrated assessment* 4.1 (2003), pp. 5–17.

[86] E. Kamsties. “Understanding Ambiguity in Requirements Engineering”. In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Springer Berlin Heidelberg, 2005, pp. 245–266.

[87] D. M. Berry. “The Inevitable Pain of Software Development: Why There Is No Silver Bullet”. In: *Radical Innovations of Software and Systems Engineering in the Future*. Ed. by M. Wirsing et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 50–74.

[88] D. J. Snowden. “Cynefin: a sense of time and space, the social ecology of knowledge management”. In: *Knowledge Horizons: The Present and the Promise of Knowledge Management*. Ed. by C. Despres and D. Chauvel. Butterworth-Heinemann, 2000.

[89] D. J. Snowden. “Multi-ontology sense making: A new simplicity in decision making”. In: *Informatics in Primary Care*. 2005.

[90] B. Johansen. *Get There Early: Sensing the Future to Compete in the Present*. Berrett-Koehler Publishers, 2008.

[91] A. Aurum and C. Wohlin. "Requirements Engineering: Setting the Context". In: *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg, 2005, pp. 11–15.

[92] A. Aurum et al. *Managing Software Engineering Knowledge*. 1st. Springer-Verlag Berlin Heidelberg, 2003.

[93] J. P. Shim et al. "Past, present, and future of decision support technology". In: *Decision support systems* 33.2 (2002), pp. 111–126.

[94] A. MacLean et al. "Questions, Options, and Criteria: Elements of Design Space Analysis". In: *Human-Computer Interaction* 6 (1991), pp. 201–250.

[95] M. Nagel. "The QUARC Metamodel: A Communication-Based Generic Project Model". PhD thesis. Technische Universität München, 2012.

[96] K. Pohl et al. *Software Product Line Engineering. Foundations, Principles, and Techniques*. Vol. 49. Springer Berlin Heidelberg, 2005.

[97] A. K. Thurimella et al. "Identifying and Exploiting the Similarities between Rationale Management and Variability Management". In: *12th International Software Product Line Conference*. 2008, pp. 99–108.

[98] A. K. Thurimella and B. Bruegge. "Issue-based variability management". In: *Information and Software Technology* 54.9 (2012), pp. 933–950.

[99] H. H. Olsson and J. Bosch. "Climbing the stairway to heaven: Evolving from agile development to continuous deployment of software". In: *Continuous Software Engineering*. Ed. by J. Bosch. Springer International Publishing, 2014, pp. 15–27.

[100] T. Garland. *The Scientific Method as an Ongoing Process*. 2015. URL: `http://idea.ucr.edu/documents/flash/scientific_method/story.htm`.

[101] A. Cho. *IBM Cloud Garage: Hypothesis-driven development*. 2013. URL: `https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development`.

[102] B. O'Reilly. *How to Implement Hypothesis-Driven Development*. 2014. URL: `https://www.thoughtworks.com/insights/articles/how-implement-hypothesis-driven-development`.

[103] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage Publications, 2002.

[104] S. Easterbrook et al. "Selecting Empirical Methods for Software Engineering Research". In: *Guide to Advanced Empirical Software Engineering*. London: Springer, 2008, pp. 285–311.

[105] R. Wieringa and A. Moralı. "Technical Action Research as a Validation Method in Information Systems Design Science". In: *Design Science Research in Information Systems. Advances in Theory and Practice*. Ed. by K. Peffers et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 220–238.

[106] F. M. van Eijnatten and G. Putnik. "Chaos, complexity, learning, and the learning organization: Towards a chaordic enterprise". In: *The Learning Organization* 11 (Dec. 2004), pp. 418–429.

[107] "Rationale and Software VV&T". In: *Rationale-Based Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 175–185.

[108] F. Auer and M. Felderer. "Current State of Research on Continuous Experimentation: A Systematic Mapping Study". In: *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Aug. 2018, pp. 335–344. ISBN: 978-1-5386-7383-6.

[109] J. L. Taylor. *Hypothesis-Driven Development*. 2011. URL: `http://www.drdobbs.com/architecture-and-design/hypothesis-driven-development/229000656`.

[110] P. N. Robillard. "Opportunistic problem solving in software engineering". In: *IEEE Software* 22.6 (Nov. 2005), pp. 60–67.

[111] J. Björk et al. "Lean Product Development in Early Stage Startups". In: *Proceedings of the From Start-ups to SaaS Conglomerate: Life Cycles of Software Products Workshop (IW-LCSP)*. 2013, pp. 19–32.

[112] D. Greer and G. Ruhe. "Software Release Planning: An Evolutionary and Iterative Approach". In: *Information and Software Technology* 46 (Mar. 2004), pp. 243–253.

[114] W. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press Cambridge, 2002.

[115] A. J. Giacomin. "Letter to the Editor: $\pi\alpha\upsilon\tau\alpha\ \rho\varepsilon\iota$: Everything flows". In: *Journal of Rheology* 59.473 (2015).

[116] F. Buschmann et al. *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. Volume 4. John Wiley & Sons, 2007.

[117] E. Evans. *Domain-Driven Design: Tackling Complexity In the Heart of Software*. Addilson-Wesley, 2003.

[118] B. Kane. *The Science of Analysis Paralysis: How Overthinking Kills Your Productivity & What You Can Do About It*. 2015. URL: `https://doist.com/blog/analysis-paralysis-productivity/`.

[119] L. Lin. *McKinsey Issue Tree Example: Problem-solving and Decision-making*. 2011. URL: `https://www.slideshare.net/interviewcoach/mckinsey-issue-tree-example`.

[120] M. Hertzum. "Problem prioritization in usability evaluation: From severity assessments toward impact on design". In: *International Journal of Human-Computer Interaction* 21.2 (2006), pp. 125–146.

[121] P. Korhonen et al. "Multiple criteria decision support-A review". In: *European Journal of Operational Research* 63.3 (1992), pp. 361–375.

[122] F. Shull et al. *Guide to advanced empirical software engineering*. Springer London, 2008.

[123] C. Grimm. "Workflow for Empirical and Opportunistic Domain Engineering in a Corporate Environment". MA thesis. Technische Universität München, 2017.

[124] S. Klepper and B. Bruegge. "Impact of Hypothesis-Driven Development on Effectiveness, Quality, and Efficiency in Innovation Projects". In: *Software Engineering und Software Management*. Gesellschaft für Informatik, 2018, pp. 181–188.

[125] W. H. Delone and E. R. McLean. "The DeLone and McLean Model of Information Systems Success: A Ten-Year Update". In: *Journal of Management Information Systems* 19.4 (2003), pp. 9–30.

[126] J. Bloomberg. *The Agile Architecture Revolution: How Cloud Computing, Rest-Based SOA, and Mobile Computing are Changing Enterprise IT*. John Wiley & Sons, 2013, pp. 257–260.

[127] P. Ralph and P. Kelly. "The dimensions of software engineering success". In: *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. 2014, pp. 24–35.

[128] C. B. Seaman. "Qualitative Methods". In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull et al. Springer, 2008, pp. 35–62.

[129] J. M. Pearson and J. Shim. "An empirical investigation into DSS structures and environments". In: *Decision Support Systems* 13.2 (1995), pp. 141–158.

[130] N. Bolloju et al. "Integrating knowledge management into enterprise environments for the next generation decision support". In: *Decision Support Systems* 33.2 (2002), pp. 163–176.

[131] S. Buckingham Shum. "Negotiating the Construction of Organisational Memories". In: *Journal of Universal Computer Science* 3 (1998), pp. 55–78.

[132] A. MacLean et al. "Design rationale: the argument behind the artifact". In: *ACM SIGCHI Bulletin* 20.SI (1989), pp. 247–252.

[133] T. C. Daniel. "Data visualization for decision support in environmental management". In: *Landscape and Urban Planning* 21.4 (1992), pp. 261–263.

[134] M. Bryce and T. Bryce. "Make or Buy Software?" In: *Journal of Systems Management* 38.8 (1987), p. 6.

[135] T. Rands. "The key role of applications software make-or-buy decisions". In: *The Journal of Strategic Information Systems* 1.4 (1992), pp. 215–223.

[136] W. H. Higaki. "Applying an improved economic model to software buy-versus-build decisions". In: *Hewlett-Packard Journal* 46.4 (1995), pp. 61–66.

[137] K. Harris. "Using an Economic Model to Tune Reuse Strategies". In: *Proceedings of the Fifth Annual Workshop on Software Reuse*. Vol. 4469. Department of Computer Science, University of Maine, Orono, ME. 1992.

[138] W. Kunz et al. *Issues as Elements of Information Systems*. Tech. rep. University of California, Berkeley, 1970.

[139] A. Widera. "Decision Support for Continuous Software Engineering in Complex Problem Domains: A Case Study". MA thesis. Technische Universität München, 2018.

[140] H. A. Simon. "Rationality as process and as product of thought". In: *The American economic review* 68.2 (1978), pp. 1–16.

[141] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice-Hall, Inc., 1995.

[142] M. Hoffmann et al. "Requirements for requirements management tools". In: *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.* IEEE. 2004, pp. 301–308.

[143] J. M. Van Bruggen et al. "A cognitive framework for cooperative problem solving with argument visualization". In: *Visualizing argumentation*. Springer, 2003, pp. 25–47.

[144] J. M. Brunetti et al. "Formal linked data visualization model". In: *Proceedings of International Conference on Information Integration and Web-based Applications & Services*. ACM. 2013, p. 309.

[145] J. Heckman. "Varieties of selection bias". In: *The American Economic Review* 80.2 (1990), p. 313.

[146] M. E. Oswald and S. Grosjean. "Confirmation Bias". In: *Cognitive illusions: A handbook on fallacies and biases in thinking, judgement and memory* 79 (2004).

[147] J. Maxwell. *Qualitative Research Design : An Interactive Approach*. Sage Publications, 2012.

[148] J. R. Feagin et al. *A Case for the Case Study*. UNC Press Books, 1991.

[149] M. B. Doar. *Practical JIRA Administration: Using Jira Effectively: Beyond the Documentation*. O'Reilly and Associates, 2011.

[150] R. K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 2003.

[151] MindTools. *Eisenhower's Urgent/Important Principle*. 2016. URL: `https://www.mindtools.com/pages/article/newHTE_91.htm`.

[152] S. McBride. *RICE: Simple prioritization for product managers*. 2016. URL: `https://www.intercom.com/blog/rice-simple-prioritization-for-product-managers/`.

[153] J. Nielsen. "Heuristic evaluation". In: *Usability inspection methods*. John Wiley & Sons, 1994, pp. 25–62.

[154] J. R. Lewis. "Usability Testing". In: *Handbook of Human Factors and Ergonomics* 12 (2006), pp. 1275–1316.

[155] Scottish Qualifications Authority (SQA). *F1VT 34: Interactive Media: Authoring: Creating a screen-based prototype*. 2007. URL: `https://www.sqa.org.uk/e-learning/IMAuthoring01CD/index.htm`.

[156] A. M. Davis. "Operational Prototyping: A New Development Approach". In: *IEEE Software* 9.5 (Sept. 1992), pp. 70–78.

[157] F. Herrera et al. "A sequential selection process in group decision making with a linguistic assessment approach". In: *International Journal of Information* 80 (1995), pp. 1–17.

[158] M. Bundschuh and C. Dekkers. *The IT measurement compendium: estimating and benchmarking success with functional size measurement*. Springer Science & Business Media, 2008.

[159] B. Peischl et al. "Constraint-based recommendation for software project effort estimation". In: *Journal of Emerging Technologies in Web Intelligence* 2.4 (2010), pp. 282–290.

[160] D. Hedgebeth. "Making use of knowledge sharing technologies". In: *Vine* 37.1 (2007), pp. 49–55.

[161] SourceForge. *FreeMind Download Statistics*. 2019. URL: `https://sourceforge.net/projects/freemind/files/stats/timeline`.