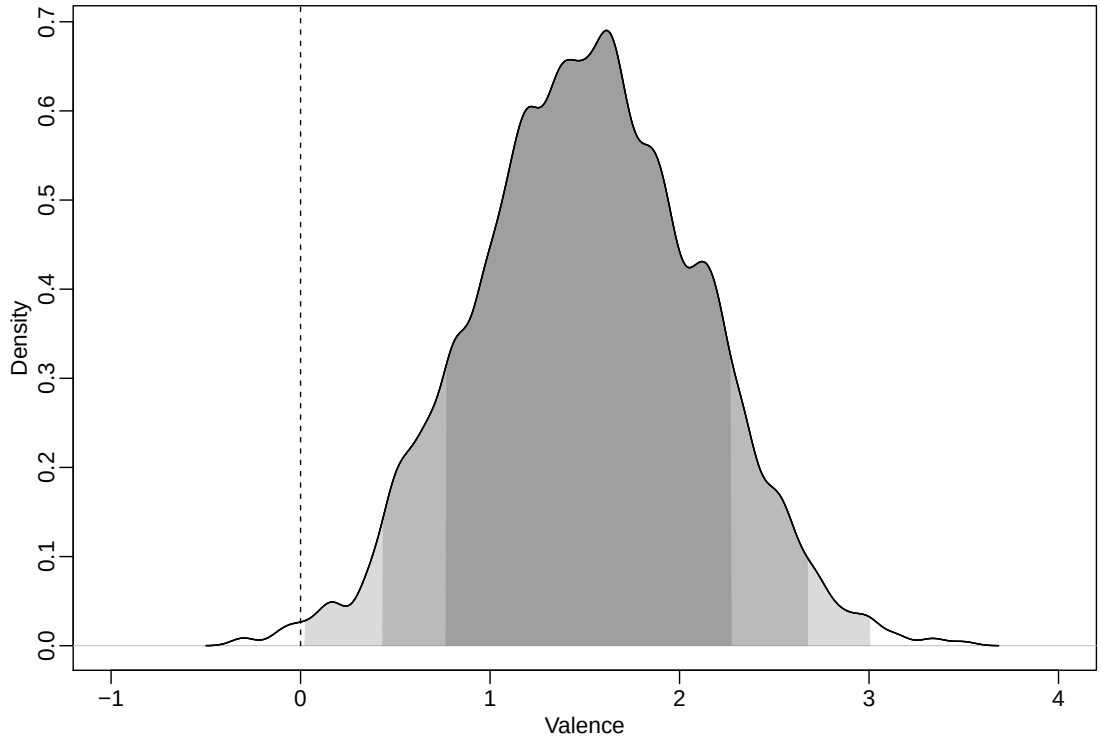


Measuring affective states from architectural technical debt

A psychoempirical software engineering experiment using the self-assessment manikin

Master's thesis in Computer Science and Engineering

JESPER OLSSON, ERIK RISFELT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Measuring affective states from architectural technical debt

A psychoempirical software engineering experiment using the self-assessment manikin

JESPER OLSSON, ERIK RISFELT

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Measuring affective states from architectural technical debt
A psychoempirical software engineering experiment using the self-assessment manikin
JESPER OLSSON, ERIK RISFELT

Supervisor: Terese Besker, Department of Computer Science and Engineering, Chalmers Universtiy of Technology
Co-supervisor: Antonio Martini, Department of Informatics, University of Oslo
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The contrast in valence for the scenario investigating the effect of cyclically-dependent modularization on the affective state of software practitioners. The shaded regions show the 99 %, 95 % and 80 % highest density intervals.

Typeset in LaTeX
Gothenburg, Sweden 2019

Measuring affective states from architectural technical debt
A psychoempirical software engineering experiment using the self-assessment manikin
JESPER OLSSON, ERIK RISFELT
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context: While the technological and financial aspects of technical debt have been extensively researched, the consequences on human aspects remain mostly uncharted. At the same time, recent psychoempirical software engineering research links the affects of software practitioners to organisational performance.
Objective: To empirically investigate the causal relationship between architectural technical debt and the affects of software practitioners.
Method: A mixed-methods approach with 40 software practitioners from 12 companies was used, combining repeated measurements laboratory experiments and semi-structured interviews.
Result: Based on a set of 200 data points, the existence of tiny tangles negatively impacts the valence of software practitioners with more than 99 % certainty. No links were found between professional background and variations in affective state. Moreover, software practitioners receive positive affects from challenging software engineering tasks and negative affects from architectural technical debt and deadlines.
Limitation: The subjects were industry professionals obtained through convenience sampling. Additionally, the treatments, albeit similar to industry code, were small and isolated examples that lacked the full spread and severity of technical debt encountered in practice.
Conclusion: By combining our results with the existing literature on psychoempirical software engineering, strong arguments can be made in favour of the hypothesis that the effects of technical debt on industry professionals carry high technological and financial risks.

Keywords: Technical debt, architectual technical debt, affective state, affects, behavioral software engineering, psychoempirical software engineering, self-assessment manikin, laboratory experiment, thematic analysis.

Acknowledgements

We would like to thank Terese Besker and Antonio Martini for supervising this thesis. Their knowledge and engagement have made it possible to achieve a thesis of higher quality than any of us had expected.

Jennifer Horkoff also deserves thanks for her work in examining this thesis. Your comments and criticism have been very helpful in bringing clarity into the study.

Next, we would like to thank Richard Torkar for informing us about the replication crisis in psychology, recommending the use of Bayesian statistics and providing valuable assistance with the quantitative analysis.

We would also like to thank Gül Calikli for introducing us to the SAM and for assisting us with the experiment design.

Finally, we would like to thank all companies and participants that have taken part in our study, as well as all students participating in our pilot studies. Without you, this thesis could never have been.

Jesper Olsson, Erik Risfelt, Göteborg, June 2019

Contents

List of Figures

List of Tables

Glossary

affective state
: An umbrella term for human emotions, feelings and moods. It is used interchangeably with affects.

affects
: An umbrella term for human emotions, feelings and moods. It is used interchangeably with affective state.

arousal
: The dimension of the PAD framework concerned with excitement. When experiencing high arousal, one feels stimulated, excited, frenzied, jittery, wide-awake, or aroused. When experiencing low arousal, one feels relaxed, calm, sluggish, dull, sleepy, or unaroused.

ATD
: Architectural Technical Debt. A type of TD that is related to architectural decisions, such as violations of best practices, consistency and integrity constraints of the architectures, or the implementation of immature architecture techniques. Consequences include immature architectural artefacts and compromised quality attributes. In this thesis ATD is not distinguished from design debt.

BSE
: Behavioural Software Engineering. It is the research field for the study of cognitive, behavioural and social aspects of software engineering performed by individuals, groups or organisations.

debt
: A concept that describes the situation where one party owes money to another party.

dominance
: The dimension of the PAD framework concerned with control. When experiencing high dominance, one feels controlling, influential, in control, important, dominant, or autonomous. When experiencing low dominance, one feels controlled, influenced, cared-for, awed, submissive, or guided.

HDI
: Highest Density Interval. A method in Bayesian statistics for reporting results. The interval describe what parts of the posterior contain the most probable values.

interest
: The fee that the debitor pays for the opportunity to use lended money. In a software engineering context, it is the extra effort induced by sub-optimal quality levels in software artefacts.

PAD
: The PAD models adhere to psychology's dimensional framework for affects and represent the entire human emotional space along the three dimensions of valence (or pleasure), arousal, and dominance.

principal
: The amount of the creditor's money that the debitor is allowed to use for the duration of the loan. In a software engineering context, it is the effort required to turn the current quality level of a software artefact into its optimal level.

PSE
: Psychoempirical Software Engineering. It is the research area for software engineering studies that use theory and measurements from psychology.

SAM
: Self-Assessment Manikin. It is an instrument from psychology for measuring affects with the PAD framework. It provides ordinal data and is administrated to the subjects for self-reporting their valence, arousal, and dominance with the help of pictorial representations.

TD
: Technical debt. In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.

valence
: The dimension of the PAD framework concerned with pleasure. When experiencing high valence, one feels happy, pleased, satisfied, contented, or hopeful. When experiencing low valence, one feels unhappy, annoyed, unsatisfied, melancholic, despaired, or bored.

1
Introduction

In today's fast evolving software landscape, there is a constant demand for new and innovative features. The delighters of yesterday may well be basic needs tomorrow. The product that does not evolve will before long be overtaken by competitors. However, is this constant evolution sustainable? It is said that to err is human, and software is developed by humans. So there are going to be errors in the software that humans develop.

Some of these errors lead to the software behaving in unintended ways. Such errors are commonly referred to as defects or bugs. However, some of these errors might not cause unintended behaviour, but are errors nonetheless. Further is the case when these errors are introduced knowingly because it is faster to code using shortcuts that enable reaching milestones faster. In this case, the errors should rather be label as trade-offs, as they shorten the time to the next feature by decreasing the overall quality in the software.

Collectively, the unintended errors and the intended trade-offs could be referred to as *sub-optimal constructs*. Although they have different origins—one from a lack of knowledge, the other from taking shortcuts—they have some similar consequences. One consequence that is widely agreed upon is that it is harder to extend and improve code that has a lot of these constructs, thereby slowing the rate at which the coveted new features can be pushed out into production. For each new feature, more sub-optimal constructs are added to the code and, unless the constructs are corrected, their numbers will keep on growing. Development time will increase, until the rate of new production crawls to a halt.

At this point the sub-optimal constructs can be ignored no longer; they have to be corrected in order to resume production. The code must be refactored. However, these constructs might not be trivial to correct. They may take a lot of time and effort to fix; time that can not be used to develop new features. This inevitably leads to the ever-present conflict of whether to improve the quality of the software or to keep pushing for new functionality. What makes the situation even worse is that the constructs rarely exist in isolation. They bleed into each other, intertwining, making their resolution even more costly to attain. The sum of the problems is greater than its parts.

In 1992, Ward Cunningham [1] coined the term Technical Debt (TD) to describe the sub-optimal constructions that did not cause unintended behaviour, yet could be confidently label as sub-optimal, nonetheless. One can see why the financial metaphor is appealing: each time a new construction is introduced, a new loan is taken, and the debt grows. Eventually, the time and effort to correct the sub-optimal constructions need to be dedicated, the debt needs to be repaid, least further

progress becomes impossible. However, this only covers the taking and repayment of the debt. There is a missing piece, the reason to repay the debt in the first place.

That piece is the additional time that it takes to develop new features when adding code on top of the sub-optimal constructs. That addition of code on top of the mistakes, as mentioned earlier, takes longer than if the constructs had not been there. The debt taken, in the form of the constructs, causes interest: extra time that needs to be devoted to working with the constructs but not fixing them.

This poses a managerial dilemma: to what extent must we compromise the ability to meet the needs of the future in order to meet the needs of the present? In theory, this question could be answered based on business goals. In practice, however, it is close to impossible to resolve the dilemma due to the numerous unknowns. These unknowns arise from many sources.

First, there are often mutual "gulfs of understanding". On the one hand, the decision makers seldom have the technical expertise required to understand underlying engineering problems. On the other, the software engineers often lack sufficient understanding of corporate strategies, stakeholder needs, and other business-related aspects.

Second, while the TD metaphor does provide grounds for common understanding, it is difficult to fully leverage. Over the years, the term has evolved and changed, and today it means different things to different people. This has slowed the incorporation of the concept into the software industry and there is a clear division between the state of the art and the state of practice.

Third, while the number of studies on the topic is rapidly growing, the field is far from saturated. One of the challenges yet to be overcome is how to accurately valorise the interest of TD. Notably, the effects of TD on the software engineers themselves has barely been investigated.

This is somewhat surprising, as we have probably all heard colleagues complain about a particularly bad piece of code that is just too difficult to understand or too difficult to extend. Is this just venting, or is there more to the complaints? Can repeated exposure to this kind of bad code cause long-term frustration and resentment?

The effects of disengaged employees have received attention in recent years, and the results are serious. Numerous organisational performance outcomes have been closely linked to the engagement of employees, including turnover, talent bleed, and customer satisfaction [2]. Within the software engineering field, close to half a hundred types of consequences have been identified in relation to the displeasure of software developers. Unhappy developers are worse at correcting defects [3] and less productive [4]. They also produce lower quality code and may straight up erase parts of the system [5].

The consequences of unhappy software practitioners are no doubt important managerial and organisational concerns. Perhaps even more so in the contemporary competitive business landscape, which is characterised by a demand for software practitioners that is far above the supply. As a result, companies are pressured to safeguard their employee retention, but may find themselves at a loss.

There certainly are many avenues that can be explored for findings ways to make, and keep, software practitioners happier. For example, are startups and Silicon Val-

ley companies renowned for their somewhat unconventional perks and work practices [6]. The direction we are looking at, however, is closer to the employees themselves and the practices co-align with the interests of the company.

In modern offices, it is commonplace to find the rooms spacious, clean and well lit. Often, there are bits of greenery. The trend is not random, nor without cost. Instead, it seems that the decision makers agree that nice offices make more productive, or perhaps happier, employees and are willing to invest resources accordingly. Yet, we argue that the software practitioner spends a large portion of their conscious awareness outside those spaces. Instead, they are immersed in the spaces of the project code base, which forms a sprawling web of numerous software artefacts and work tasks.

In an ideal world, those webs would be well-curated and organised so that minimal extra effort is required to perform one's assignments. In reality, this is not the case. The typical web is a tangled mess that haphazardly knits threads that significantly slows down the work. Industry professionals estimate that 36 % of their development time is wasted because of the existence of this mess [7]. This is worth repeating: an estimate of 36 % of all development time is going down the drain because of the interest of TD.

The effect of this mess, of the software practitioners' virtual work environment, on their feelings, job satisfaction, and engagement has yet to be researched. The matter is doubtless very complex and multifaceted, but also very important to elucidate. As outlined above, the human aspects of software engineering carry substantial risks when not properly managed. Those risks need to be incorporated in the equation determining how much TD is a sustainable amount.

This is where this thesis comes in. We will investigate TD with the goal of uncovering potential consequences related to human aspects. Because of the size of the topic, we delimit our scope to examine how TD interacts with the feelings, or affective states, of software practitioners. Therefore the research objective of this thesis is defined as follows.

Research Objective: The goal of this thesis is to investigate the relationship between TD and affective state from the point of view of software practitioners.

Our results provide strong evidence in favour of TD impacting the affective state of software practitioners. With more than 99 % certainty, we were able to show that the occurrence of even small circular dependencies, also known as tiny tangles, made practitioners feel more unhappy, unsatisfied or melancholic. The participants further validated this result by reporting that while the code was similar to what one encounters in industry, the TD of the scenarios was more manageable than situations in practice.

Further, we were unable to find any conclusive evidence that professional characteristics, such as work experience or role, would be related to the affective state induced in software practitioners. Notably, these findings run in counter to the beliefs of developers [8].

We also found that software practitioners experience changes to their affective state, both positive and negative, as part of their daily work. These changes are

experienced quite frequently; as often as weekly for some practitioners. Although most of these experiences are positive, several negative experiences were also reported, together with consequences as severe as burn-out.

The remainder of this thesis is divided into five sections. The next section introduces background information relevant to the research objective. In the third section, we describe the mixed-methods approach used for investigating the issue. The fourth section presents the results from the quantitative and qualitative methods. Section five discusses the findings and lists threats to validity. The last section, six, concludes the thesis.

2
Background

This section presents the topics necessary to understand the problems investigated in this thesis. We begin with introducing the research areas of technical debt and behaviour software engineering, before the objective of the study is positioned in relation to the existing body of knowledge. The section is concluded with a short description of the quantitative and qualitative approaches that will be used for analysing the results.

2.1 Technical Debt

Technical debt (TD) is a framework for reasoning about various aspects of software quality. It was conceptualised by Ward Cunningham in 1992 as a financial metaphor for explaining the technical reasons for how already shipped code can slow down future development, unless it is reworked as new domain knowledge is gained [1].

Since its coinage, the debt metaphor has surged in popularity. Similar to a game of Chinese whispers, the original meaning has been changed over the years [9]. It has been misinterpreted, diluted and reduced, but also extended, evolved and systematised. The concept is today so overloaded that it can mean widely different things to different individuals.

To some, TD is invokable as a justification for poor software craftsmanship [10]. Others distinguish TD from other quality issues due to being an intentional trade of long-term benefits for short-term ones [11]. In academia, a resolution was proposed in 2016. During the Dagstuhl Seminar 16162, a new definition for TD was derived, which has been adopted in this thesis.

> "In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability." [12, p. 112]

This definition was accompanied by a conceptual model of the contemporary understanding of TD. For the sake of completeness, the model is presented in its entirety in Figure 2.1, even though only the consequences of TD are of interest for this thesis. It is noteworthy that none of those consequences are concerned with

human aspects[1], much less with the feelings of software practitioners.

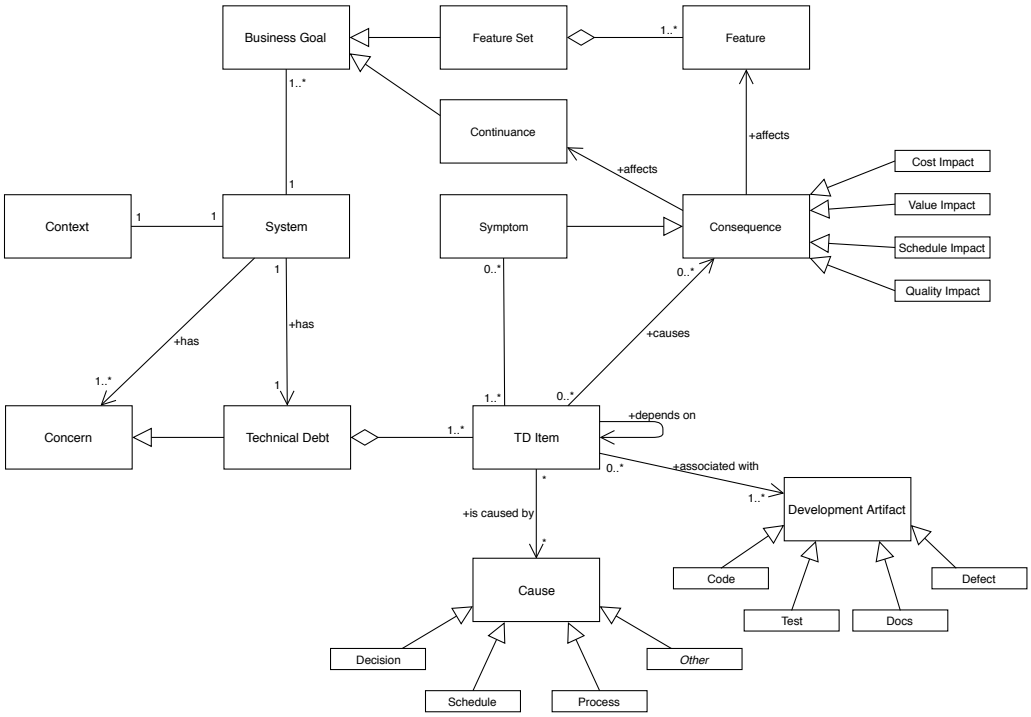


Figure 2.1: The conceptual model of technical debt drafted during the Dagstuhl Seminar 16162.

There are several plausible explanations for the absence of human aspects in the model. It could be that there exists convincing evidence suggesting that TD does not cause variations in such areas. However, as we will detail in Section 2.3, this is not the case. It appears, instead, that this category is missing because the topic has been neglected in the TD research. This indicates a gap in the body of knowledge that this thesis could occupy.

2.1.1 Financial Terminology

At its heart, TD is a framework that provides an additional facet for communication of software quality. As such, it provides software engineers the opportunity to reflect upon their work in from a complimentary perspective. Perhaps more important, however, is its possibility for translating, sometimes quite deep, technical understandings into a language that non-technical personnel are more likely to understand. Because the target language is that of finance, TD is particularly useful for advertising long-term technical risks to stakeholders with a financial background, such as managers and financial backers.

While powerful when successfully used, the framework carries risks in terms of misunderstands and mistranslations. In the same fashion, as financial stakeholders often lack deep technical competence, software practitioners frequently lack financial

[1]Note that the consequence of value impact refers to monetary worth, as opposed to principles and standards of behaviour.

expertise. In a systematic literature review, Ampatzoglou et al. found that the TD research was inconsistent and infrequent in its use of financial approaches [13]. As a result, they synthesised a glossary of the financial terms connected to technical debt. While those definitions are adopted in this thesis, the focus of this study is not on the financial aspects of TD and we will hence simply briefly present the basic concepts of debt, principal and interest.

The concept of debt is at the center of TD and conceptualises how much financial funds a debitor (or borrower) owes a creditor (or lender) [13]. Typically, an individual becomes indebted because they require more money than they can dispense with and must loan the difference.

The loan specifics an agreement between the debitor and the creditor. The creditor temporarily gifts financial funds to the debitor for an agreed upon period of time. In return, the debitor obliges to reimburse the creditor with a larger sum at the end of the period [13].

Another way to phrase the loan is that the borrower agrees to compromise their ability to meet their future needs in order to meet those of the present. The borrower acknowledges that they make a loss on the loan, in the hopes that the extra money will allow them to make decisions that will earn them more money during the period, thereby compensating the loss.

The principal of the loan denotes how much money the debitor originally borrowed from the creditor [13]. As noted above, the principal constitutes only a portion of the sum that the debitor has to repay. The remaining sum is called the interest and can be thought of as the cost for the debitor ,and profit for the creditor, of agreeing to the loan.

In a software engineering context, the principal of a TD item is formulated as the effort required to turn the current quality of its associated development artefact into its optimal level [13]. The interest is thus the extra effort, for example during maintenance, induced by this difference [13].

2.1.2 Taxonomy

Several attempts have been made to classify TD along various dimensions, such as reckless versus prudent [14], and deliberate (or intentional) versus inadvertent (or unintentional) [14, 15]. Additionally, various researchers have classified TD according to type [16, 17, 18].

One of those types is Architectural Technical Debt (ATD), which is concerned with sub-optimal decisions regarding the software architecture [9] and constitutes an essential element of the overall TD [19]. It is a type of TD that is related to architectural decisions, such as violations of best practices, consistency and integrity constraints of the architectures, or the implementation of immature architecture techniques [19]. Consequences include immature architectural artefacts and compromised quality attributes [19].

As far as we can tell, no consensus has been reached in the research field with regards to how ATD is related to design debt. Some classifications merge them [16], while others separate the two [17, 18]. However, this separation is not clear-cut and does not, consequently, provide disjunct sets [18, 20]. In this thesis, we have

adopted the former stance and will use ATD as an umbrella term for TD concerned with software design and architecture.

2.1.3 Software Smells

Software smells are indicators of software quality issues that could potentially impair the maintainability and evolvability of software systems [21]. At the same time, software smells are not necessarily definite quality problems, as the system context might justify their existence [21].

Architectural smells are software smells that occur at a high level of abstraction and include inappropriate design solutions and violations of design principles [22]. Similar to ATD, terminological consensus has not been achieved. While most authors seem to agree that architectural smells are on a higher level of abstraction than code smells, the relationship to design smells remain unclear. Some studies differentiate solely between architectural and code smells, thus leaving design smells unaddressed [22]. Others suggest that design smells could be referred to micro-architectural smells [23], indicating that the are similar, but of different scale. On the other hand, this distinction is not evident as can be seen that, for example, Cyclic Dependencies and Hub-Like Dependencies are considered design smells [23] or architectural smells [24], depending on study.

In this thesis, design smells are distinguished from architectural smells based on scale and connectivity. Design smells are small, local smells that exist in an isolated part of the software system, while architectural smells are wide-spread.

2.2 Behavioural Software Engineering

There is a growing body of literature that recognises the importance of interdisciplinary research between software engineering and psychology [25]. It is a well-known fact in both academia and industry that software engineering tasks are human activities and, thus, impacted by human aspects [26, 27, 28, 29]. Perhaps one of the most striking examples is the attention that agile software development research has received since the manifesto[2] was written [25, 30].

However, much of the research on human aspects has up until recently has been more or less isolated. There was no common platform for this kind of research [30] until Lenberg, Feldt, and Wallgren proposed a new research area in 2015, in an attempt to clarify the studies concerned with human aspects of software engineering [30]. They named this area Behavioural Software Engineering (BSE) and defined it as "the study of cognitive, behavioral and social aspects of software engineering performed by individuals, groups or organizations." [30, p. 18]

Despite an extensive list of 55 concepts comprised in BSE [30], the field has not yet solidified. The founders acknowledged that the initial classification may not be complete and expect more concepts to be incorporated into BSE. One example of this is the concept of affective state, which is a commonly used umbrella term for feelings, emotions and moods [31]. Because it is explicitly stated that emotions are

[2]The agile manifesto is documented at `http://agilemanifesto.org/`

included in the cognitive part of BSE [30], we argue that research on affective state and, by extension, the field of psychoempirical software engineering and this thesis should be considered part of BSE.

2.2.1 Psychoempirical Software Engineering

In 2015, Graziotin, Wang, and Abrahamsson proposed a new research area, Psychoempirical Software Engineering (PSE), in order to unify the software engineering studies using proper theory and measurements from psychology [31]. In the same paper, they synthesised psychology theory concerned with affects (see Section 2.2.1.1) and provided guidelines for conducting PSE research on the topic [31]. Since this thesis investigates affects, it is founded on those guidelines and should be categorised as PSE research.

There is no unified psychological framework that matches all possible research objectives concerned with affective states [31]. Instead, researchers are encouraged to select an appropriate framework depending on the objectives of their study. The research objective stated in Section 1 implies that this thesis intends to measure how TD (the stimulus) influences the affective state of software practitioners (the participants). For this purpose, the PSE guidelines recommends using the dimensional framework or, more specifically, the PAD models and names the Self-Assessment Manikin (see Section 2.2.1.2) as the most appropriate measurement instrument [31].

2.2.1.1 Affects

Affects (or affective states) is an umbrella term for human emotions, feelings and moods [31] that will be employed in this thesis. This term is used in accordance to the PSE guidelines, for standardisation purposes. It should be emphasised that the feelings-related concepts sometimes are distinguished and nuanced to an extent that is outside the scope of this thesis. Although the field of psychology has not reached consensus on the definitions of those terms [31], certain sources define an emotion as a reaction to external stimuli [31], while a mood is a prolonged emotional state of mind, for which there is no obvious stimuli [31].

As mentioned in Section 2.2.1, there exist multiple psychological frameworks for affects, but the one of interest for this study is the dimensional framework. The framework defines affects in terms of several, clearly distinct, emotional dimensions. One of these is the PAD models, which will be used for this thesis.

In the PAD models, three dimensions are identified[32]. These are on the pleasure-displeasure (valence), arousal–nonarousal (arousal) and dominance–submissiveness (dominance) scales [31]. Valence expresses how much attraction one finds to a stimulus [31]. Arousal corresponds to how mentally awake and reactive one feels in reaction to a stimulus [31]. Dominance represents how challenging a task is a perceived in relation to one's skills.

In more ordinary terms, valence is the dimension of happy versus unhappy [33]. When valence is high, one feels happiness, satisfaction, contentment, or hopefulness, whereas low valence is expressed in feeling unhappiness, annoyance, dissatisfaction, melancholy, despair, or boredom [33]. Similarly, arousal corresponds to the dimension of excited versus calm [33]. Arousal is high when one feels stimulated, excited,

frenzied, jittery, wide-awake or aroused, while one feels relaxed, calm, sluggish, dull, sleepy or unaroused when arousal is low [33]. Finally, dominance refers to the dimension of controlled versus in-control [33]. A high dominance reflects feeling controlling, influential, in control, important, dominant or autonomous, and low dominance are feelings characterised as controlled, influenced, cared-for, awed, submissive or guided [33].

It should be emphasised that affects equate to neither motivation nor job satisfaction [31, 34], but the concepts are related. Affects can influence these, as well as other complex constructs [35].

2.2.1.2 The Self-Assessment Manikin

The Self-Assessment Manikin (SAM) is an instrument for measuring affective state within the PAD framework [36, 37, 38] and is a recommended tool for conducting PSE research [31]. As can be seen in Figure 2.2, the SAM comprises three horizontal scales. These are pictorial representations of the three dimensions that PAD uses to express affects [31, 36].

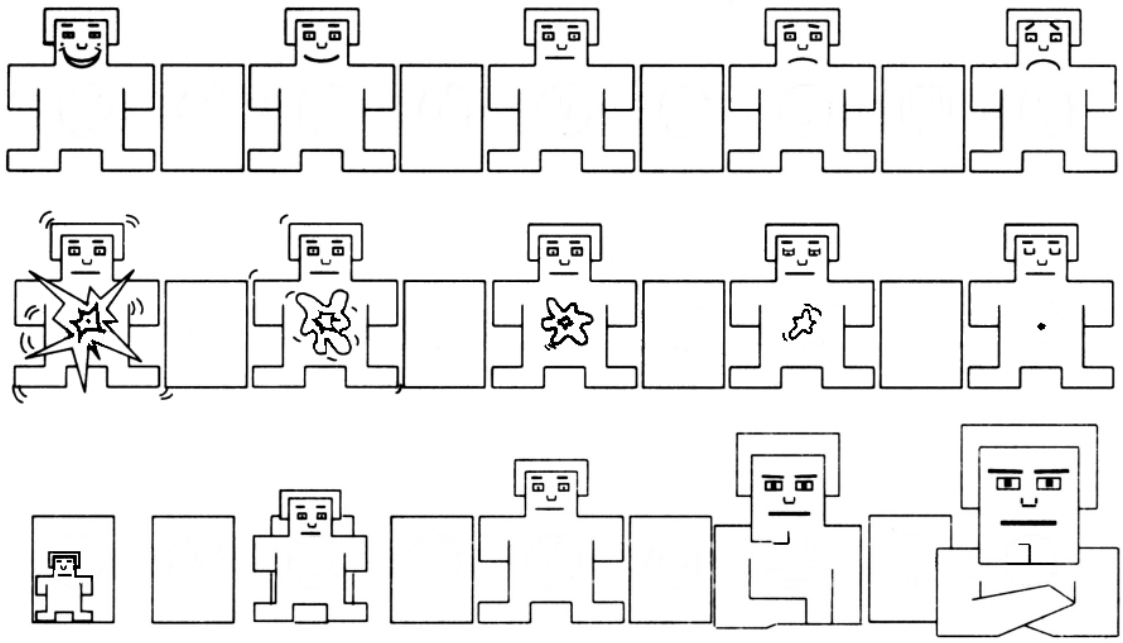


Figure 2.2: The SAM measurement instrument. SELF ASSESSMENT MANIKIN © Peter J. Lang 1994

The top row shows valence by letting the manikin's facial expression range from a smile (high valence) to a frown (low valence). The middle row illustrates arousal through the manikin being full of energy (high arousal) or being still, with a sleepy facial expression (low arousal). The bottom row demonstrates dominance by changing the relative size of the manikin from small (low dominance) to large and adopting an assertive body language, with crossed arms (high dominance).

Each scale is of the ordinal type, with nine-point granularity. This means that, while ratings have a direction, it is not possible to infer the magnitude of change

between points. For example, the change between the first and the second points could be higher than between the sixth and the seventh.

Because the SAM relies on self-reporting, the scores are not standardised according to objective points of reference. Although individuals are consistent with themselves [31], the ratings cannot be assumed to be consistent between individuals [31]. In other words could two individuals rate the same affective state in two different ways. Consequently, investigations administrating the SAM should following a within-subject (or repeated measures) design [31].

The pictorial design of the SAM has many benefits. The lack of verbal components means that the SAM can be administered to a broader range of the population, including individuals with a non-English mother tongue or language disorders, and children [37]. Additionally, the SAM can measure direct affective reactions as it can be filled out in a short amount of time [39] and eliminates cognitive processing [38]. In combination with the stylised characters, as opposed to photographs of humans, the SAM is less susceptible to many types of biases [39] and can be considered culture independent [39].

The SAM is an established instrument in psychology. It was developed by Lang in 1980 [36], who obtained a copyright for it in 1994. The SAM [36] and instructions for correct administration [33] is available for use for academic, not-for-profit research[3]. Since its creation, the SAM has been subject to extensive validation research [39] and used in numerous studies [39, 40].

In recent years, the SAM has received criticism in recent years. It has been argued that the SAM may suffer from biases because of its age. Supposedly, the SAM is founded on outdated design principles [40] and is no longer intuitively understood by participants, even when correctly administered [40]. In spite of this criticism, the SAM will be the measurement instrument of this study, as per the recommendation of the PSE guidelines.

2.2.2 The Replication Crisis

A central issue in contemporary psychology research is the validity of the existing body of knowledge as well as of new findings. In 2012, the editors of Perspectives on Psychological Science published a special issue dedicated to the so-called "replication crisis" in psychology and other research fields, in response to a number of controversies concerning the reproducibility of studies [41].

There is now much evidence to support the claim of severe replicability issues in psychology. In addition to the discourse that led up to the special issue, the Open Science Collaboration conducted a large-scale examination of the reproducibility of psychology research. They replicated 100 papers published in journals central to the field and obtained positive results in 35 instances despite the expected number being approximately 89 instances [42].

This crisis is an important issue that has damaged the credibility of the entire research field [41]. Replication is one of the core aspects of research validity and the scientific method itself; if the purpose of research is learning about the world, truths

[3]Information about how to obtain the SAM can be found at `https://csea.phhp.ufl.edu/Media.html`

should be distinguishable from fabrication or misconceptions. This discrimination is compromised when the validity of the discoveries is under significant threats, e.g. not being independently verifiable. As an example of the severity of these issues, there are known instances of founding theories that cannot be replicated [43].

It is generally accepted that the replication crisis should be combated by reforming several facets of the research field, which concerns the entire range from research teams to journals. Many potential causal factors for the crisis have been suggested and include over-interpretation of results, reliance on Null Hypothesis Significance Testing (NHST), fraud, ineffectual incentive scheme and publication bias [41, 43].

Especially the reliance on NHST has been a target for criticism and several researchers advocate a transition to Bayesian statistics. NHST is particularly problematic in noisy contexts where the phenomena under study are small and highly variable [43]. Additionally, NHST is an unsuitable method for inference when there are competing hypotheses [44, 45, 46].

Beyond the technical aspects of the suitability of statistical approaches, there are a number of issues in how the research community judge significance. A stance that has been widely adopted, by journal editors and several hundred scientists in various fields [47], is to stop determining the significance of research based on preconceived thresholds, e.g. the commonly used $p < 0.05$. Such practices introduce a wide range of biases, ranging from self-censorship to misinterpretations of data, to p-hacking [43, 41, 44, 46].

2.3 Related Work

The effect of TD has been studied extensively in recent years. At least six secondary studies [16, 17, 13, 18, 48, 19] on the topic were published between the years 2012 and 2018 and there is now much evidence to support the hypothesis that TD often is detrimental to software companies and may cause palpable negative consequences for software longevity.

In spite of the attention paid to the field, there appears to be few studies conducted on the topic of human aspects, which is reflected in Figure 2.1, encountered earlier in this section. Of the meta studies mentioned above, only one indicated that the occurrence of TD may impact the software practitioners working with the afflicted software product. More specifically, it was suggested that TD could have a negative impact on morale in the long term and, by extension, increase turnover rates [16]. However, the cited sources are blog posts, without apparent scientific grounds for the claims.

This appears to be the situation in general. In our investigation of the body of knowledge, little research on the topic was found. Disregarding opinion-based pieces and studies that simply cite the literature, only a few publications were uncovered. While providing some empirical evidence in support of the hypothesis that TD has a negative impact on the software practitioner's psychology [16, 49, 50], the findings are excerpts of what individual practitioners reported during interviews. Those findings seem opportunistic rather than the result of planned research questions, which naturally threatens their validity.

By the end of the literature review, only two studies with the intent to empirically investigate the effects of TD on human aspects had been found. Spínola et al. surveyed software practitioners about TD folklore and found medium to high consensus among software practitioners that TD is related to their morale [51]. This relationship was also detailed by Ghanbari et al., whom conducted interviews and an online survey in order to determine what the effects of occurrence and management of TD are on developers' morale [8]. Their findings show that the existence of TD negatively impacts morale, but also that morale is increased from proper TD management.

As we have previously noted (see Section 2.2.1.1), complex cognitive constructs such are morale are different, but related, to affects. This implies that the objective of this thesis is novel, but also that the study could overlap and synergise with the findings of the studies mentioned above. Indeed, Ghanbari et al. help map out the way of this thesis by reporting promising leads. First, their result indicate that software practitioners have split opinions about certain types of TD, while other types, such as ATD, are considered very important [8]. Second, software practitioners held beliefs that could not be supported by the data, as shown by the incorrect notion that TD induced affects depend on the level of work experience [8]. Specifically, evidence suggests that work experience does not impact the scariness or frustration of TD-heavy artefacts such as legacy code [8].

From the side of BSE research, a recent study provide evidence that TD lowers the valence of software practitioners. The study employed a survey to investigate causes for unhappiness of software practitioners and found that being stuck in problem-solving, poor code quality and coding practices were among the most significant causes [52]. While these findings are intriguing, the study also calls for additional research for further understanding the causes, especially those concerned with software artefacts.

This thesis will answer the call by investigating the issue of bad code quality in the context of ATD research. As such, there is some overlap between the studies, but also important differences. The other study used a survey in order to investigate what situations developers think cause unhappiness. This thesis will, among other things, employ (see Sec) laboratory experiments to investigate this claim.

2.4 Bayesian Statistics

The issue of statistical inference has been a controversial and disputed subject within several research fields (see Section 2.2.2). One of the discourses is the debate of Bayesian or frequentist statistics. In some unfortunate instances, the debaters segregate into different camps that uphold the belief that the method they advocate is superior. While our standpoint towards the argumentation in the field at large is neutral, we have chosen to use Bayesian statistics in this study. We view frequentist and Bayesian statistics as separate sets of methods, each with their own pros and cons. In some situations, the most suitable method is frequentist, in others it is Bayesian. Problems arise when the researcher selects a method that is unsuitable for their data. Or worse, misapplies the method itself. Against this background, our rationale for choosing Bayesian is founded on the recommendations of combating

the ongoing replication crisis in psychology (see Section 2.2.2).

Since Bayesian statistics is currently not widely adopted in the research field of software engineering, we will not make the assumption that our readers are familiar with the subject. Because of this, we dedicate the rest of this section to introduce the basics of Bayesian statistics. The aim is merely to provide sufficient theory to make our data analysis transparent and more understandable. For more detailed discussions about Bayesian statistics in the large, we refer to other sources[4].

2.4.1 Conceptual Overview

Bayesian methods make inferences based on three distinct components: a generative model, a set of priors and a set of data. Generative models are functions that express how likely data are to occur. Because of this, they are sometimes referred to as likelihood functions [45]. Likelihood functions come in many forms, including common probability distributions. This provides a convenient conceptualisation of a generative model: supposing that the model is a standard normal distribution, it would tell us that observing data equal to one is more likely than observing data equal to two (roughly 25 % and 5 %, respectively).

Continuing with the same example, it is a sound approach to question why the selected likelihood function was a standard normal distribution. This is, after all, no more than a special case of normal distributions where the mean is zero and the standard deviation is one. Other values of these parameters would have given a different answer to the question of how likely some data are to occur. For example, a normal distribution with mean three and the same standard deviation would have swapped the likelihood (i.e. roughly 5 % and 25 % for observing data equal to one and two, respectively), while one with a mean of one hundred and a standard deviation of two would have claimed that the likelihood of observing either value is minuscule (roughly 10^{-500} %). Similarly, the output of the generative models would change if it were some other probability distribution than the normal.

As can be seen, choosing a suitable model is essential and the more data one has, the more important does the choice become. It is, however, not necessary to specify the parameters in the way illustrated here. Instead of fixed values, the parameters can be described with uncertainty. That is to say, the parameters may themselves be probability distributions. This is quite natural, if one contemplates the idea that a singular value also is a probability distribution. Granted, it is a distribution that the value will occur 100 % of the time and any other value will occur 0 % of the time, but it is still a distribution. This illustrates how the use of distributions in the conventional sense is a less draconian way of specifying the parameters.

Regardless of how the parameters are specified, they must be determined. This is where priors come in. A prior is a specification for the parameters of the generative model. As such, they are explicit beliefs about the phenomena under study and can be used to incorporate previously known evidence into the generative model. For example, previous research may point towards the mean following a particular normal distribution. Then the prior for that parameter can be set to that distribution.

[4]Well-reputed textbooks include *Statistical Rethinking*, *Bayesian Methods for Hackers* and *Think Bayes*

Similarly, one can reflect a lack of previous evidence by using conservative priors, such as uninformative priors or weakly informative priors [45].

Finally, the data set retains its ordinary meaning: some empirical evidence about the underlying truth of the phenomena under study. Typically, the data is collected by researchers investigating a particular problem.

Conceptually, Bayesian inference is carried out by constructing a generative model for representing the problem, incorporating previous knowledge in the priors. Next, two steps are repeated for a large number of iterations. First, the priors are sampled in order to obtain one set of values for the generative model's parameters. Second, the model generates simulated data [45], which is tagged with the values of the parameters.

After these steps, the simulated data will be compared to the proper data and some of the simulated data will be a good match. By analysing the tags, we will typically find that several sets of parameter values can give rise to matching data. However, some tags will exist in larger proportions than others. That is, of all the simulated data that match the proper data, some parameter values are more likely to produce the data that was collected.

At this stage, one could determine that the set of parameter values that corresponded to the largest proportion of matching data are the best ones. However, this would reduce the evidence to a single answer, a point estimate, and thereby remove the uncertainty that was quantified during the procedure. By reporting all the proportions of the parameter sets, a probability distribution for those parameters is obtained, which provides a more nuanced description. This distribution is known as the posterior distribution.

2.4.2 Interpreting Results

Reality is often complex and empirical investigations seldom provide universal truths [45]. Data can be interpreted in many ways and the results are often dependent on context. This situation is reflected in the output of Bayesian models. Rather than providing point estimates, which is common in many types of statistical tests, the posterior is a distribution of many possible values, of which some typically are more probable than others. As such, the analysis retains much information that otherwise would be obscured and can therefore be understood in relation to different contexts.

One common way of interpreting the posterior distribution is through the use of Highest Density Intervals (HDIs). These intervals report what parts of the posterior contain the most probable values [45]. For example, the 95 % HDI show what values are the 95 % most probable, which reflects the threshold of $p < 0.05$ that is commonly used in research.

However, there are issues in relying solely on one such threshold (see Section 2.2.2). Rather than using them to determine whether or not the results are significant, it can be more useful to disclose the posterior together with several HDIs at different levels [45], presenting the results with more nuance.

2.5 Thematic Analysis

Thematic analysis is an approach for analysing qualitative data and is frequently applied in psychology [53] and also in software engineering [54]. In this thesis, Braun and Clarke's guidelines of [53] will be used.

Thematic analyses are conducted in six phases, during which the researcher familiarise themselves with the data and codes the data for themes to search for, before those themes are then reviewed, defined and reported [53]. The approach can be applied to many different situations, as there are many variations to the method. Braun and Clarke list three dimensions along which thematic analyses vary.

First, they can be inductive or deductive, which refers to in what manner the the data is analysed and corresponds to the decision of bottom-up or top-down investigations, respectively [53].

Second, the method can be conducted on an explicit or interpretative level. The former focuses on the surface of the phenomena, while the latter extends to underlying constructs [53].

Third, one can analyse the data with the realist or constructionist methods. The realist, or essentialist, method reports the reality of the participant and their experiences, whereas the constructionist method investigates their cultural background and societal constructs [53].

3
Methods

This thesis is concerned with human aspects of ATD. The research on this topic is limited (see Section 2.3) and our investigation could be considered novel. Consequently, we have chosen to apply a mixed-methods approach with both quantitative and qualitative data, in order to obtain a stronger understanding of the phenomena. The first method is a laboratory experiment, which is conducted in order to understand the potential causality between ATD and affective state through quantitative data. The second is a thematic analysis of semi-structured interviews, which aims to explore peripheral and connected issues. In total, 40 participants from 12 companies took part in the study.

3.1 Study Design

Each participant took part in both the experiment and the interview in the same session. Conceptually, the session was divided into three parts, as illustrated in Figure 3.1. First was the pre-task instructions (see Section 3.6), which served to provide the participant with sufficient context to understand the aim of the study and inform them about how their data would be handled and their rights. Secondly, the measurement sitting (see Section 3.7) served to collect quantitative data on the causal effect of ATD on affective state. Finally, the post-task interview (see Section 3.8) served to collect data about the participant's background profile and qualitative data on peripheral issues.

Each session was bounded to a maximum of 90 minutes. The time estimate for the different session parts were 10–20 minutes for the pre-task instructions, 40 minutes for the measurement sitting, and 30–40 minutes for the post-task interview. These estimates were based on pilot studies and were set to include some float time for any unforeseen events.

Every part of the session comprises, in turn, sub-parts. Those sub-parts will only be briefly introduced now, as they are further detailed in later sections. The pre-task interview is conceptually composed of three sub-parts: a confidentiality assurance, SAM instructions, and a task description. The confidentiality assurance addresses ethical concerns of the research and ensures informed consent. The SAM instructions informs the participant about how to use the measurement instrument, while the task description explains what the participant will do during the experiment.

The measurement sitting also constitutes three parts: an anchor point, deacclimatisation periods, and scenarios. Because the experiment is a repeated measures design, its validity is threatened by learning effects. The anchor point attempts to

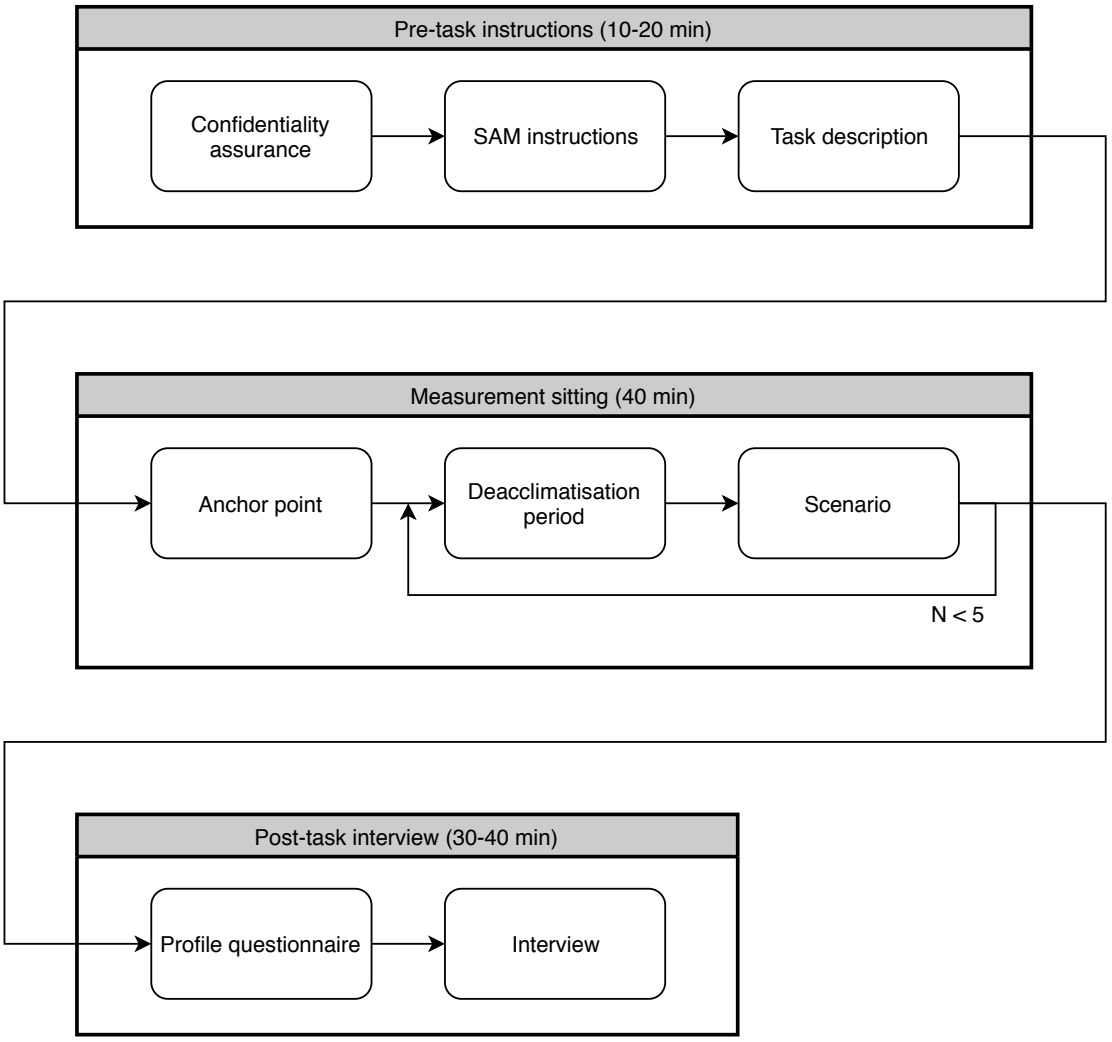


Figure 3.1: A conceptual model of the sessions. The participants attended a 90-minutes session, composed of three parts and eight sub-parts.

mitigate those through the use of a common practice measurement. Between each measure is a deacclimatisation period in order to mitigate the risk of the participant's affects carrying over between the measures. Each measure is a scenario that contains a code example with either a high (H) or a low (L) amount of ATD.

The post-task interview comprises two parts: a profile questionnaire and an interview. The profile questionnaire is a short questionnaire about the participants' professional experience with software. Finally, the interview is a semi-structured interview about how the participant perceived the study and their perception of code maintainability and feelings in general.

The experimental material used throughout the sessions is listed in Table 3.1. Each piece of material is further detailed in later sections and appendices as shown in the table. Like the conceptual components of the sessions, the material is included here solely for overview.

Table 3.1: The experimental material that was used in the experiment.

Item	Section	Appendix
Pre-task Manuscript	3.6.1.1	A
Confidentiality Agreement	3.6.1.2	B
Anonymous ID	3.6.1.3	-
Name-ID List	3.6.1.4	-
SAM Instructions	3.6.1.5	C
SAM Rating Booklet	3.6.1.6	D
SAM Example Ratings	3.6.1.7	E
Task Description	3.6.1.8	C
Scenarios	3.7.1.4	F
Post-task Manuscript	3.8.1.1	G
Participant Professional Profile Questionnaire	3.8.1.2	H

3.2 Goals

As mentioned in Section 1, the objective of this study is to investigate the relationship between TD and affective state from the point of view of software practitioners. Therefore, we will now define the following three main research questions (RQs) and the six related sub-questions.

RQ1: How does ATD impact the affects of software practitioners?
RQ1.1: Does ATD, in terms of design smells, cause variations in software practitioners' affects?
RQ2: How do professional characteristics correlate with changes in affects due to software smells?
RQ2.1: Does formal education correlate with changes in affects due to design smells?
RQ2.2: Does work experience correlate with changes in affects due to design smells?

RQ2.3: Does work context correlate with changes in affects due to design smells?
RQ3: Do software practitioners attribute changes in affects to software engineering tasks?
RQ3.1: Do software practitioners readily recall instances where they have experienced changes in their affective state due to software engineering tasks?
RQ3.2: Do software practitioners frequently experience affective state variations as a consequence of software engineering tasks?

The first research question, RQ1, seeks to address the relevance and importance of ATD to the affective state of software practitioners. The first sub-question, RQ1.1, narrows the scope to analysing causal relationships between design smells and affects.

The second research question, RQ2, concerns the question of whether or not there are individual differences that influence how software smells are perceived. Due to confidentiality concerns, we delimit ourselves to solely investigating professional characteristics. RQ2.1 will provide an understanding of whether or not higher formal and relevant education is related to how software smells are experienced. For example, it might be the case that individuals with more relevant education get more annoyed with software smells due to a more extensive theoretical knowledge base. On the other hand, it seems equally possible that more educated individuals may be more carefree towards smells, due to an increased likelihood of identifying suitable resolutions. Similarly, RQ2.2 will reveal the same aspects from the perspective of practical experience, while RQ2.3 examines the aspects based on the work role and the language the participant reports as having the most experience in.

The last research question, RQ3, aims to investigate whether software practitioners have previously experienced changes in their affective state because of the software artefacts that they work with. Sub-question RQ3.1 focuses on exploring how well-established the connection between software engineering tasks and affects are, in the minds of the respondents. Finally, RQ3.2, concentrates on gauging the frequency and systematics of the affects to assess the risk of long-term issues.

3.3 Participants

This study was conducted between 2019-03-26 and 2019-04-17 at 12 companies, of which one was a government agency, selected through convenience sampling. The researchers used their own professional networks as well as the networks of the student union and the university in order to establish contact with companies interested in taking part in the study. The researchers approached the companies by pitching the study—e.g. at student career fairs or through personal communication—and asked them to provide participants. Naturally, some companies were uninterested or reported that they could not spare the resources. Additionally, some companies that were initially interested withdrew before the study commenced.

The companies who did partake in the study are listed in Table 3.2. Note that one of the companies, Skatteverket, is a government agency rather than a for-profit corporation. Further note that two of the companies asked to remain anonymous.

The total number of participants was 40 and the contribution of each company ranged between 1 and 7, with an average of 3.33 and a standard deviation of 1.87.

Each company is shortly described in Section 3.3.3.

Table 3.2: The 12 companies that took part in the study.

Company
Anonymous 1
Anonymous 2
Burt
Greenbyte
IFS
PRI Pensionsgaranti
Pulsen
Saab
Semcon
Skatteverket
Stratsys
Wireless Car

The variation in the number of participants is an effect of the procedure. Because we wanted to reduce the risk of discriminating certain types of companies, we did not set any lower limit to the number of participants. Instead, each company was asked to contribute as many participants as they were comfortable with.

Similarly, we did not want to discriminate certain types of roles. Consequently, we informed the companies that anyone who could be considered a software practitioner was eligible for participation. However, we disallowed participants who could have personal bias with the researchers, for example former colleagues or (former) fellow students of the researchers.

3.3.1 Participant Motivation

Several ethical issues in research are concerned with the motivation for participants to take part in the study. This section aims to disclose how those were addressed in this thesis. We will describe how the following concepts were handled: commitment, consent, confidentiality, and motivation.

The commitment from the participant could be considered relatively minor, since they were not required to spend more than up to 90 minutes of their regular work time in order to participate. Because the company selected their own participants (see Section 3.3), the participants were presumably not subjected—by their manager nor other colleagues—to adverse effects. Additionally, each participant was informed that they could decline to answer any questions or withdraw from the study at any time. In other words, there was no coercion from the researchers.

Informed consent was obtained by the participant signing a confidentiality agreement. The agreement discloses the purpose of the study and promises anonymity and, additionally, the destruction of the participant's data upon their request. The confidentiality agreement can be found in its entirety in Appendix B.

Confidentiality was ensured through the use of anonymous identifiers. The full procedure is detailed in Section 3.6.2.

Participant motivation was induced without the researchers providing any monetary or other direct benefits for participating in the study. Instead, voluntary participation was encouraged by the use of informed consent and anonymity, as described above.

3.3.2 Allocation to Treatments

One of the methods employed in this study was a laboratory experiment. Experiments are typically conducted in order to investigate causality, rather than correlation, by measuring if the phenomena changes as the factor under consideration is altered. In our case, we want to investigate if the affects induced by software scenarios change when there is more ATD.

The experiment follows a repeated-measures design. This means that the participants is subjected to the situation several times. In this study, this translates to each participant being subjected to different software scenarios. Consequently, there is another level of complexity to the treatment allocation. Without repeated-measures, the participants could be divided into two groups, where one group receives the treatment with the lower level of ATD and the other receives the treatment with the higher level of ATD. When using repeated-measures, the participants are instead subjected to a series of higher or lower levels of ATD, which we will refer to as level patterns.

In this study, two level patterns were selected for use in the experiment. These were the treatments that the participants could receive. As a result, the participants were divided into two equal groups. To emphasise, these treatments determined what version of the software scenarios the participants received, but not the order in which the scenarios were administrated. The order of the scenarios was randomised, to mitigate threats to validity.

3.3.3 Company Descriptions

In this section, short descriptions of a subset of the companies are provided. These descriptions were obtained by enquiring the companies. As such, the descriptions were collected through personal communication and is the words of the companies. Omitted companies either did not provide a description or asked to not be included. The intent of the descriptions is to convey the profiles of the different companies to help understand the sample.

All offices visited as part of this study were located in Sweden, across four different cities: Gothenburg, Mölndal, Stockholm, and one undisclosed location (as that company asked to remain anonymous).

3.3.3.1 Burt

> "Burt creates a Marketing Intelligence product for the Advertising Industry."

3.3.3.2 Greenbyte

"Greenbyte develops the industry-leading renewable energy data management software Energy Cloud – used globally by wind and solar energy professionals to capture the full potential of renewable energy projects. Power plants are remotely connected to Greenbyte's platform via on-site SCADA systems – resulting in a single powerful tool to monitor, analyze, plan and control diverse renewable energy portfolios of any size. Predict, the advanced AI predictive maintenance system was added to Greenbyte's renewable energy management software portfolio, with outstanding results. Currently 17 GW of renewable energy across 5 continents and over 40 countries are monitored in Greenbyte's software. Greenbyte was founded in 2010 by 4 Chalmers alumni – since then, the company has widened the scope of products and customer base, and yearly makes the list of Sweden's most innovative companies. In 2019, over 70 professionals based in Gothenburg, Sweden and 4 other locations in Europe and the US collaborate daily to contribute to 70% annual growth."

3.3.3.3 PRI Pensionsgaranti

"PRI Pensionsgaranti, mutual insurance company guarantees and administers book reserve method pension schemes. Since 1961, SEK 300 billion have remained in, and been put to work for, Swedish companies which have chosen to manage ITP 2 using the book reserve method.

PRI has 101 employees in Stockholm and Gothenburg. We work primarily with collectively contracted pension plans, but also with companies' own plans and book reserve method early retirement pensions. In addition to credit insurance, our services include pension administration, foundation services and calculation of pension liabilities in accordance with Swedish and international accounting standards.

We develop our main insurance-systems by ourselves. We use in most cases Microsoft software .net C#.

The group also includes the subsidiaries PRI Pensionstjänst AB and PRI Stiftelsetjänst AB."

3.3.3.4 Saab

"Saab serves the global market of governments, authorities and corporations with products, services and solutions ranging from military defence to civil security."

3.3.3.5 Stratsys

"We develop smart digital tools for collaboration on strategical plans. Making the sum greater than its parts.

Stratsys is a cloud based tool for efficient strategic planning that helps every user to understand their contribution to the bigger picture. Hassle-free and smooth, as work should be. Our tools streamline work

for hundreds of thousands of people, helping them save time and to use it more wisely. As an ever-evolving and improving platform, Stratsys is the spine of business planning. The holy grail that gathers and engages all users in the same goals."

3.3.3.6 Wireless Car

"We drive the automotive industry beyond connectivity, into the future. Today.

With the ambition to lead the automotive industry beyond connectivity and into the digital society, WirelessCar is the world's leading innovator of Digital Vehicle services that connect cars, eco systems, business flows and fleets. Founded in 1999, WirelessCar has continuously built on its heritage and knowledge, and is today a highly recognized and award winning company, connecting more than four million vehicles in over 75 countries.

WirelessCar is owned by Volkswagen Group (majority shareholder) and Volvo Group (minority shareholder) and operates globally, with offices in Sweden, North America and China. Our more than 400 dedicated and highly skilled members of staff are committed to deliver world class services that exceed customer expectations and needs, bringing our partners and customers into the future of digital services. This allow our customers such as Volkswagen, Jaguar Land Rover, Daimler, Nissan and Volvo Cars to leverage the full value of connected services across the entire spectrum of always being connected."

3.4 Pilot Studies

In order to refine the procedure and identify possible problems, pilot studies were conducted as part of the experiment design. This meant that the design was improved upon, in an iterative fashion, based on issues encountered during the pilot studies.

We conducted one series of pilot studies of two different types. The first type was exploratory, in the sense that it aimed to discern trade-offs in different design choices and their feasibility. The second was confirmatory, in the sense that it served as a trial run of the design resulting from the exploratory type.

Only one series of pilot studies were conducted because no issues were identified in the confirmatory studies. This series was held prior to the data collection of the study.

The participants of these studies were students enrolled in the Software Engineering education at Chalmers University of Technology, i.e. individuals who were currently undertaking the Software Engineering bachelor[1] or master[2] programmes.

[1] `https://www.chalmers.se/sv/utbildning/program-pa-grundniva/Sidor/Informationsteknik.aspx`

[2] `https://www.chalmers.se/en/education/programmes/masters-info/Pages/Software-Engineering-and-Technology.aspx`

However, there is a trade-off in using students as subjects for the pilot studies. On the one hand, there was an ample supply of students interested in contributing to this study. They were available on short notice and using their time does not cause as much economic harm to a third party as e.g. industry professionals might cause to their company. On the other hand, students differ, in general, from industry professionals and the population of students is therefore not necessarily representative of the population under study. Further, the use of a student's time may cause more personal economic harm than that of an industry professional. Students' income is typically lower and paid on the basis of output (i.e. passing course exams) rather than work time, which managers might consider participation to allow for.

At the time of commencing the pilot studies, the number of confirmed participants was too low for us to deem it justifiable to use some of them as subjects in the pilot study. When the decision was made, several companies had not determined whether or not to take part in the study or, more specifically, how many participants they would contribute with. Therefore, the decision was made to use students as participants in the pilot study.

The data collected during the pilot studies were not included in the results of this study. The reason is twofold: the data set had a different purpose from the one collected during the main study and the populations are not necessarily representative of each other. Consequently, an analysis of the pilot study would be complex and the presentation of it might obscure the results of the main findings. Instead, we have chosen to make the pilot study data available in Appendix J but that data will not be the subject of any further analysis in this thesis.

3.4.1 Exploratory Pilot Studies

The exploratory pilot studies were conducted on a total of six students in four sessions. The goal of these studies was to evaluate different design alternatives by investigating how the participants responded to different parameters. In Table 3.3 the parameters are listed together with how they were varied and what value was decided upon after analysing the outcome.

Table 3.3: The parameters that were investigated during the exploratory pilot studies.

Parameter	Variations	Decision
Bounded scenario length	Yes/No	Yes
Class separation	Yes/No	Yes
Syntax highlighting	Yes/No	Yes
Code examples per participant	3–5	6
Simultaneous participants	1–2	1
Deacclimatisation length	0:14–1:27	2:00
Scenario length	2:48–7:00	5:00

These studies differed from the main study in two ways. First, the design did not include the profile questionnaire, which was a construct that was conceived between

the exploratory and confirmatory pilot studies. Second, the subjects were asked to participate for an estimated time of 60 minutes, instead of the full 90 minutes of the main study. This choice was made in order to mitigate the potential economic harm caused to the participants, thus reducing the risk of skewing the sample of the pilot study.

The results of pilot studies showed that participants do not require much time to size up the scenarios. Consequently, it was decided that the scenarios be time-box to 5 minutes each. This avoided the case where participants finish the task prematurely, in spite of there being design issues they hadn't addressed. Time-boxing also serves to turn the participant's task into an incompletable task: rather than being able to "finish", the participant is asked to do as much as possible in the allotted time.

Further, it was clear that there were learning effects in the experiment, but also that those were mainly concerned with the first scenario encountered by a participant. Therefore, it was decided to turn a scenario into a dummy scenario, referred to as the anchor point. The anchor point would serve to let the participants learn the procedure of the experiment and the correct usage of the measurement instrument, without introducing bias in the analysis. From the participant's point of view, the anchor point would be no different from the other scenarios, but it would not be part of the analysis. The anchor point would be the first scenario encountered and would be the same for each participant.

Some of the code examples were perceived as less suitable for the experiment than others and the decision was made to remove them. The low version of one scenario was very small and may have influenced the ratings of the participants, especially if they are able to correctly identify it as a low example, potentially skewing the results of the study. It also seemed to confuse the participants of the pilot study, when it was not contrasted with the high version. This may have been due to the low version not exhibiting any apparent quality issues. The high version of the same scenario was made into the scenario used as anchor point (see Section 3.7.1.2).

One scenario was also removed, as the design smell seemed to be easy to miss, unless one knows what to look for.

Finally, it was clear that simultaneous participants influenced each other. Therefore, the experiment was only conducted with one participant at a time.

3.4.2 Confirmatory Pilot Studies

The confirmatory pilot studies were conducted on two students in two sessions. The goal of these studies was to identify potential operational problems in the procedure. As such, the procedure did not differ from the main study in any way. The experiment design that was investigated had incorporated responses to the issues identified in the exploratory pilot studies.

No problems with the experiment design were discovered in these pilot studies. They further showed that the study could be completed in full within the 90 minutes time constraint, but also that there are substantial individual differences in terms of how much time was used for the open-ended questions. Because of this, no need to alter any part of the study was identified and the design was declared final.

3.5 Setting

The study was conducted in conference or meeting rooms at the participant's company. Each session had exactly one participant. In the event that a company contributed with multiple participants, those were scheduled for different sessions. One seat was prepared for the participant, prior to their arrival. A stack of 10 blank A4 papers were placed in front of their seat, together with a pen and a fresh copy of the SAM rating booklet (see Section 3.6.1.6). The researchers' seats were next to each other, opposite the participant.

We allocated a time slot of 30 minutes before the start of each session, in order to have ample time for making these and other preparations, such as double-checking recording devices. This time would also provide buffers in the event that participants were scheduled in sequence and the prior arrived late.

3.6 Pre-task Instructions

Conducting research with human participants can be perilous, in terms of validity threats, compared to programmable experimental units. The space for misinterpretations, misunderstandings, and biases is large and could influence the result of the study [55]. One of the ways we attempted to address these issues was through the pre-task instructions.

The instructions aimed to achieve a number of goals. Not only should they provide the participant with all information needed to be part of the study, both technical and ethical. They should also acquaint the participant with the situation, so that they are less likely to commit environment-induced mistakes.

Because this study investigates affects, these phenomena become more discernible; the results would be invalid if the data is distorted due to the participant feeling nervous or threatened during some parts of the session.

In other words, the pre-task instructions does not exist in order to collect any data, nor answer any of the research questions. Instead, the instructions exist to mitigate certain threats to validity and to reduce the risk of misconduct.

3.6.1 Material

This section gives an account of all the material and equipment used during the pre-task instructions. The aim is to provide as much detail as necessary for independent researchers to understand the material and how it should be administrated. As such, this section describes characteristics of the material that might impact results.

3.6.1.1 Pre-task Manuscript

The pre-task instructions contains much information that participants should receive. In order to ensure that no essential part was excluded, we composed a memory aid in the form of a manuscript. The text is available in its entirety in Appendix A. During the sessions, a laminated copy of the manuscript was used.

3.6.1.2 Confidentiality Agreement

After the initial instructions, the participants were asked to read an sign a Confidentiality Agreement (see Appendix B). With this, the participants were informed of how their data was used in the study, that they can decline to answer any question, and that they could withdraw from the study at any time. It also contains the contact information of the researchers, so that the participants know how to contact the researchers. Finally, the agreement had the contact information of the researcher's supervisors, with concerns or complaints regarding the methods used in the study, should any such concerns arise.

3.6.1.3 Anonymous ID

In order to honour our promise of guaranteed anonymity, we used unique identifiers for every participant. Each identifier was a random sequence of alphanumerical characters. More precisely, each character was uniformly drawn with replacement from a pool consisting of digits between 0-9 and all lower and upper case letters between A-Z.

The researchers did not themselves assign identifiers to the participants. Instead, each participant was asked to draw an identifier from a collection. As detailed in Section 3.6.1.4, the sole way to link an identifier to a participant was by means of a single, physical document.

All data collected from the participants were marked solely with their identifier. Furthermore, the link was physically separated from the data. In order to avoid confusing the researchers, the identifiers had a length of 5 for pilot studies and of 8 for regular participants.

3.6.1.4 Name-ID List

The Name-ID list was a physical list of all participants of the study and their respective anonymous IDs. The list was kept separate from all other participant information and data, in order to ensure anonymity while still allowing the researchers the option to trace specific answers back to a participant.

This tracing was required to allow the participants the option of retracting their responses from the study. Additionally, the researchers could use this list to examine possible outliers in the data set.

The rationale for using a physical list was to ensure that no residual information was kept after its destruction and that no unauthorised party would be able to access it.

3.6.1.5 SAM Instructions

The PSE guidelines highlights that the instructions for the measurement instrument can influence the affects of the participants [31]. They advise the researchers to base their instructions on any instructions accompanying the measurement instrument. Such instructions are included for the SAM in a white paper written by Lang and Bradley [33]. The guidelines also encourage publishing the instructions used in the study. In accordance with this advice, the SAM instructions used in this study are

available in Appendix C. Note that the SAM instructions and the task description (see Section 3.6.1.8) are physically part of the same document.

For the experiment, the instructions were recorded to an audio file, which was played back to the participants. This ensured that all participants received identical instructions. The audio file is available upon request.

In order to adopt Lang and Bradley's instructions for the SAM, adjustments were made in order to avoid confusing, or in other ways influence, our participants. The original instructions were written for a study that had a different design from ours. For example, that study was designed for groups of 8 to 25 individuals that rated pictures shown on a projector [33], in contrast to this study where a single participant is asked to review printed code examples.

Some changes were made to the original instructions. The administrative parts of the instructions were not compatible with our assurance of confidentiality and were therefore removed. In addition, we cut down on the amount of text that restated how to rate the dimensions, as the participants in our pilot studies reported feeling patronised.

3.6.1.6 SAM Rating Booklet

The technical report for administrating SAM specifically mentions the use of a rating booklet [33], but does not provide copies of the same. Consequently, a SAM rating booklet was constructed for this study, and is available in Appendix D.

The SAM rating booklet comprises two A4 pages that were folded into A5 pages and stapled together. The booklet only contains the SAM and the corresponding page numbers, as well as a cover page and a blank back. It could, for example, have included written instructions for how to use the tool. However, the choice was made to rely on the audio instructions as well as the first scenario (anchor point) to serve as sufficient instructions. This enabled us to make the booklet very sparse in order to distract as little as possible from the actual task of the experiment.

3.6.1.7 SAM Example Ratings

In the exploratory pilot studies, some participants reported feeling some confusion with how SAM should be used. Therefore, as a complement to the audio instructions (see Section 3.6.1.5), three images were printed and laminated to show examples of how SAM should be used. These images are available in Appendix E.

3.6.1.8 Task Description

In order to guide the participants' interactions with the scenarios, they were given a task (see Appendix C) for them to consider while inspecting the scenarios. The task was used so that the participants would not just passively look at the scenario, but instead be encouraged to engage with, and look for potential issues in, the code design.

3.6.2 Procedure

Upon arrival, the participant was greeted, thanked for their participation, and asked to seat themselves. Next, the study was introduced, in accordance with the pre-task manuscript (see Section 3.6.1.1). In order to make the participant feel comfortable with the study, the text was not played from an audio recording nor read from directly. Instead, the researchers had the document in front of themselves and read the contents in a more conversational fashion. This allowed for ad-hoc changes or clarifications as needed, while ensuring that all information was presented to the participant.

As part of the pre-task instructions, the participant was handed two copies of the confidentiality agreement (see Section 3.6.1.2) and asked to sign at least one of them. A signed copy was retrieved by the researchers and the participant was asked to draw an anonymous identifier (see Section 3.6.1.3) from a bag and hand it back to the researchers. The participant's name and anonymous identifier on a separate list (see Section 3.6.1.4). The signed copy of the confidentiality agreement and the list were stored by the researchers for safekeeping, while the remaining copy of the confidentiality agreement was left for the participant to keep or throw away at their own discretion. In the event that the participant chose not to sign the agreement, they were thanked for their interest and the experiment was terminated.

After the pre-task manuscript was finished, the researchers played back an audio file containing the SAM instructions (see Section 3.6.1.5) and the task description (see Section 3.6.1.8). At the points indicated in the instructions, the researchers placed the appropriate SAM example rating (see Section 3.6.1.7) in front of the participant. After the recording had finished, the researchers once more asked if everything was clear and clarified as needed.

3.7 Measurement Sitting

This section documents the procedure that was used to perform the laboratory experiment. It should serve as a recipe for the experiment and aims to provide all information that is necessary for independent researchers to replicate it. Furthermore, this section details choices that were made and the underlying rationale, allowing the reader to identify and evaluate threats to validity.

3.7.1 Design

This experiment is concerned with answering RQ1 and will employ the SAM (see Section 2.2.1.2) as the measurement instrument. For this reason, the experiment will follow a within-subjects design with ATD as the stimulus. Because ATD is an abstract concept, we have used five software engineering scenarios (see Section 3.7.1.4) as proxies. Each scenario exists in two versions: one exhibiting a higher (H) amount of ATD than the lower (L), refactored, version.

The experiment design is illustrated in Figure 3.2. As can be seen, the experiment occurs between the pre-task instructions and the post-task interview. The participants were equally divided among two treatment patterns (see Section 3.7.1.1).

Next, the participant was subjected to an anchor point scenario, followed by a deacclimatisation period, during which they had the option to receive clarifications about the experiment. For the rest of the experiment, the participant would inspect and rate the five scenarios, with deacclimatisation periods in between. This repetition is omitted from the figure and is instead represented by the three dots in the middle of the diagram.

Which version of each scenario the participant would rate was determined by their treatment pattern. For example, the first scenario that a participant with the LHHLH pattern encountered would be the refactored version.

3.7.1.1 Treatment Patterns

The two treatment patterns, LHHLH and HLLHL, were constructed in an attempt to mitigate biases related to the order in which the ATD levels were encountered. It is possible that some patterns would influence the participants' ratings. For example, it would be difficult to determine if ATD impacts the participant's affects if the pattern was LLHHH; as any variations could simply be due to the participant becoming e.g. more bored or comfortable with the experiment. Some patterns, such as LHLHL, could also potentially be recognised by the participant, allowing them more opportunities to influence the data. Because of such issues, we not only constructed patterns with less inherent bias, but that were also each others' compliments.

3.7.1.2 Anchor Point

The anchor point served to help calibrate the participants. Although the participants received instructions for their task and how to rate their affects using the SAM, those instructions might not have been internalised before the start of the first scenario. By providing the anchor point as a practice measure, misunderstandings can be corrected without influencing the data. In order to further reduce bias, the anchor point would be common for all participants.

Similar to the scenarios (see Section 3.7.1.4), the anchor point is a software engineering scenario that the participants would rate with the SAM. As a matter of fact, there should not be a difference between the anchor point and the scenarios, from the viewpoint of the participant. What makes the anchor point unique is that it is an artefact for mitigating learning effects.

3.7.1.3 Deacclimatisation Periods

There is evidence that affective states may persist after the stimulus has been removed [56]. Consequently, any affects induced in a prior measure might carry over to subsequent ones. While such biases are partially mitigated by randomising the order of the scenarios, it was decided to also introduce short pauses of 2 minutes between every scenario. These pauses are referred to as deacclimatisation periods.

We were unable to identify suitable tasks that the participants could perform during the deacclimatisation periods to help them reset their affects. Our investigation found that some studies have used relaxing documentary films for this purpose

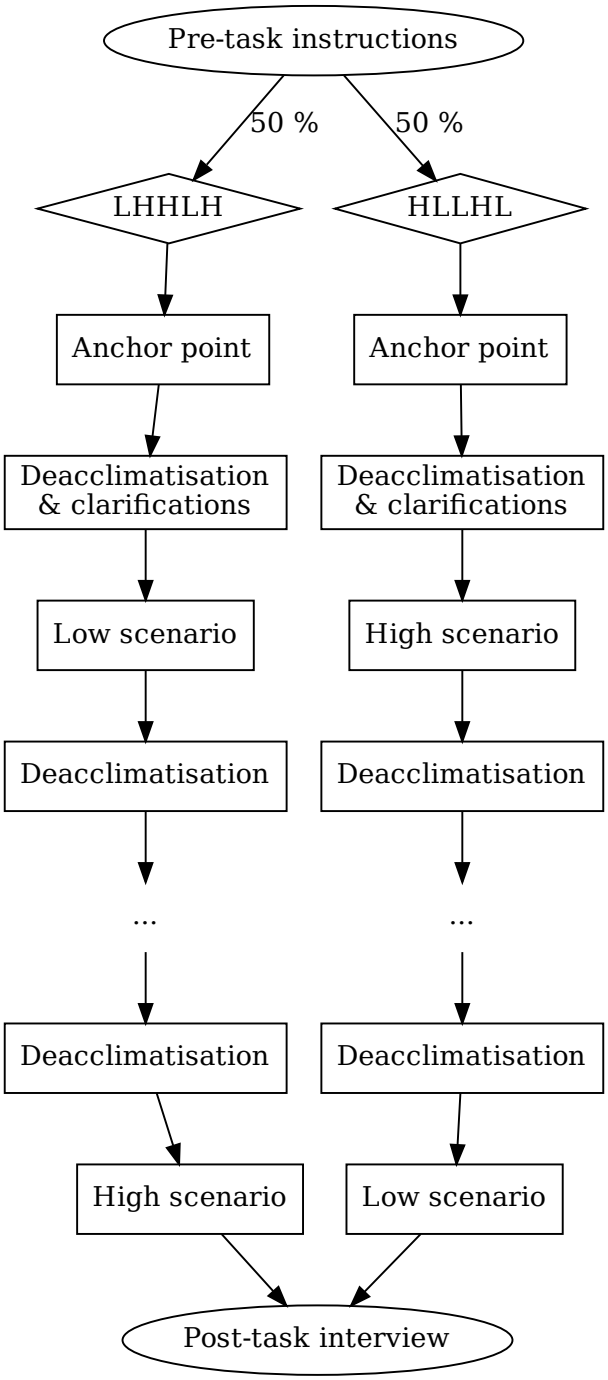


Figure 3.2: The design of the measurement sitting.

[57], but neither a concrete procedure nor the material was referenced. In the end, we were unable to obtain detailed guidelines before commencing the data collection phase. Because of this, we settled on a silent pause of non-activity.

The length of the pauses was derived from our pilot studies (see Section 3.4.1). Initially, a pause time of 90 seconds was attempted and deemed insufficient. Subsequently, 120 seconds were tried and the pilot participants provided positive feedback.

3.7.1.4 Scenarios

The scenarios were constructed in order to serve as concrete instances of ATD. The scenarios were composed of a code example written in Java and a software engineering task that the participant should perform on that code. Furthermore, each scenario existed in two variants. One variant (H) exhibited a higher degree on technical debt, by containing a known software quality issue, while the other variant (L) was its refactored equivalent. It should be noted that the scenarios could exhibit additional issues that remained unaltered between the versions. This is further elaborated in Sections 3.7.2.1-3.7.2.6.

For each scenario, the participant was given 5 minutes to perform the software engineering task. This limit was set in order to ensure that the experiment did not exceed the allocated time of 90 minutes, but also to reduce the risk of certain types of biases. For example, if no lower time bound was provided, the participant might speed through the scenario in an attempt to appear more competent in the eyes of the researchers. This could cause them to overlook important aspects of the scenarios. Additionally, if the participant is allowed to determine when they have finished the scenario, their affects might be influenced due to the sensation of completing a task. By letting the lower and upper bounds coincide, such effects should be reduced, as there was no room to complete the task.

The length of the scenarios was inferred from the pilot studies. In the earlier pilots, the participants were allowed to complete the tasks at their own pace. From those, it was noticed that there were some individual differences, but also that 5 minutes seemed like an appropriate length. This amount of time was deemed sufficient for the participants to comprehend the core issues with the code examples, but also did not appear to be long enough to make the participants think their time was being wasted.

In order to reduce threats to construct validity, we chose not to construct the scenarios ourselves, as this might have introduced bias. Instead, we investigated the literature for concrete examples of ATD. With concrete representations, we mean examples embodying the debt and associated solutions in diagrams or source code. Concrete representations also differ from the abstract representations often found in existing catalogues [22]. Rather than serving as a pattern of identification, the examples should be realistic problems that could be encountered in practice. Ideally, such examples should be understandable to software practitioners, regardless of domain or previous experiences. Additionally, it might be insufficient to reference software smells, since smells are not necessarily indicative of definite quality problems [21].

From our investigation, it seems that there is a deficit in concrete representations of ATD in the existing body of knowledge. The examples we found in scientific

publications did not fulfil our criteria or were in other ways unsuitable for this experiment. For example, some instances would be too large for the participants to gain an understanding of in the allotted time. Similarly, some were rather domain specific or could be considered outside the knowledge sphere of the common practitioner. Such factors might well dominate the affects of the participant to such an extent that potential variations induced by ATD would become too small to measure. In others, smells had been eliminated without it being evident if the solutions improved the overall quality of the system. With the caveat that our investigation was neither systematic nor exhaustive, we thus call for research on synthesising definite architectural quality issues in concrete representations.

In the end, the scenarios were based on the contents of [23]. Although the source is not a scientific publication, the examples it provides seemed suitable for our purposes, at least after minor modifications (see Section 3.7.2).

3.7.2 Materials

The only new material introduced in the measurement sitting was the code examples that constituted the anchor point and the scenarios. The examples were all were based on the contents of [23], which is a book on software design smells and appropriate refactorings. The examples were selected according to the following scheme. First, of all the 25 smells listed, those that fulfilled the following criteria were retained.

1) The smell was listed as negatively impacting the understandability quality attribute;
2) The example had a concrete representation; and
3) The smell was deemed suitable for the experiment context, i.e. the quality issue was explicit, the smell was estimated to being able to stimulate affective state variations, and the smell could be understood in a short time.

In some instances, the examples were deemed unsuitable for the experiment even though the smells passed the criteria. For instance, the example could be too domain specific to be deemed sufficiently accessible. Those were modified by altering names and context, while still retaining the core design issue.

In order to standardise these examples, they were all represented in Java source code. Because some of the scenarios were provided solely as software diagrams, we were forced to transform those to the best of our ability. In addition, we tried to reduce bias toward coding styles by ensuring that the source code followed standard style guides. More precisely, we adopted Google's Java code style[3] and enforced it with the software tool *Checkstyle*[4]. We also ensured that the code examples contained no syntactic errors by extending it with necessary elements, which were omitted from the example after successful compilation. The code was rendered into

[3]A comprehensive description is available at `https://google.github.io/styleguide/javaguide.html`

[4]Publically available at `http://checkstyle.sourceforge.net/`

image format with Eclipse[5] using the standard syntax highlighting of Eclipse. These images were printed in full colour and each class cut out separately and laminated.

After this process, the remaining examples were tested in pilot studies, and the six best were retained for the scenarios and the anchor point. Those scenarios are listed in Table 3.4 together with the software smell they embody. Each code example is further detailed in Sections 3.7.2.1-3.7.2.6. All code examples, in all levels, are also presented in Appendix F.

Table 3.4: The six scenarios used in the experiment and the smells they embody.

ID	Smell	Smell category
Anchor	Imperative Abstraction	Abstraction smell
ScA	Missing Encapsulation	Encapsulation smell
ScB	Missing Hierarchy	Hierarchy smell
ScC	Broken Modularization	Modularization smell
ScD	Cyclically-Dependent Modularization	Modularization smell
ScE	Rebellious Hierarchy	Hierarchy smell

In addition to all smells being listed as negatively impacting the understandability quality attribute, the smells also have several other impacted quality attributes in common. The quality attribute that each scenario impacts [23] is listed in Table 3.5.

3.7.2.1 Anchor Point Scenario

The anchor point scenario is an instance of software designs where an object-oriented programming language is used, but without adhering to the object-oriented paradigm [23]. Instead of an object encapsulating its capabilities, its operations are decomposed into other classes.

In addition to the design smell, the code example had a couple more quality issues. First, the `copy` method has two arguments. This indicates that the method does not create a fresh instance of a `Report`, but instead overwrites one of the arguments with the internals of the other. From the parameter names, it is unclear which report is the original and which is the copy.

Furthermore, the source material does not specify the relationship between the `display` method and the `Report` class. This relationship could be a hidden field, set in a constructor, but it is also possible there is no relationship in the source code. For example, the report can be accessed from a specific location on the file system or be missing altogether.

Because this code example is the anchor point, it was decided that the latter alternative should be used. The rationale was that this approach hopefully would cement in the participants the notion that some issues are not necessarily obvious and that there may be more aspects to the code design than is apparent at a glance.

[5]Eclipse IDE is available at `https://www.eclipse.org/`

Table 3.5: Software quality attributes affected by each scenario.

ID	Understandability	Changeability	Extensibility	Reuseability	Testability	Reliability
Anchor	x	x	x	x	x	
ScA	x	x	x	x		
ScB	x	x	x	x	x	x
ScC	x	x	x	x	x	x
ScD	x	x	x	x	x	x
ScE	x	x	x	x		x

3.7.2.2 Scenario ScA

This scenario is an example of two orthogonal concerns that are not adequately separated, which can lead to an exponential increase in classes [23]. In this case, it was decided to change the context of the code examples, as we considered encryption too specialised knowledge for this experiment. Instead, we chose to use different types of geometric shapes and image formats. While this might compromise the realism of the examples, this was deemed as an acceptable trade-off.

3.7.2.3 Scenario ScB

This scenario embodies a design issue of underutilised polymorphism [23]. Instead of using an abstraction to represent the intent, the code relies on the checking of types at runtime.

Although the source material chose to elide parts that were deemed irrelevant to the example, a reference to the original code was included[6]. In order to extend the example with more context, the decision was made to use the original source. This enabled the full implementation of the method and the class to be provided. In addition, we could show that this was an inner class of a much larger body of code. It should be noted that the comments that point out that the original developers were aware of the design being sub-optimal were removed.

Besides this quality issue, the example exhibits more room for improvement. First, the `getBorderInsets` has the side effect of modifying one of its parameters, which is not clear from its signature. Second, there are code clones at the end of the method that could be removed by introducing a new method with a higher level of abstraction in the `Insets` class. Related to this issue is that the fields of the `Insets` class are exposed to, and can be manipulated by, the clients.

3.7.2.4 Scenario ScC

This scenario showcases the phenomena where data and associated operations are inappropriately separated [23]. Within the object-oriented paradigm, data and operations are commonly seen as the same unit.

Other issues with the code example include that all implementation details have public scope and a lack of self-documenting code. None of the parameters to any of the methods in the scenario describes what they represent.

Furthermore, there are missing abstractions. An ID is likely more constrained than a `String`, for example having to be unique and adhere to a certain format. Similarly, one could also consider turning paths into an abstraction. Finally, the primitive wrapper class `Boolean` is used, without a clear rationale, over the corresponding primitive type.

3.7.2.5 Scenario ScD

This scenario illustrates the problem of co-dependent classes [23]. Such classes must typically be viewed and treated as one single unit.

[6]The full qualified name of the class is `java.swing.plaf.windows.XPStyle.java` and is part of Oracle's Java Development Kit

Further, the example uses the `double` type to represent money, which makes the code susceptible to floating point rounding errors. In addition, the `Order` does not properly encapsulate its internals, as the `items` variable can be manipulated by clients. For example, `Item`s can be added and removed.

3.7.2.6 Scenario ScE

This scenario represents a situation where subclasses do not conform to the contract of its ancestors [23]. This can cause unexpected behaviour if the client operates on an abstraction level that is higher than concrete classes.

As was the case in Section 3.7.2.2, call sites were considered to be too specialised knowledge for this experiment. As such, the decision was made to change the context of this code example as well. Instead, different types of documents were chosen as the domain of the scenario. While this might compromise the realism of the examples, it was considered more important that the participant did not get any adverse feelings due to unfamiliarity with the domain.

Besides the issue of subclasses breaking the contract of its ancestor, it is unclear what it means in the domain that a document is, for example, read-only. In some domains or situations saving and deleting such documents might be allowed, while all operations except reading might be prohibited in other domains.

Lastly, the purpose of a tagged document is ambiguous. It may, for example, represent a document whose content is tagged for different renderings, such as in an HTML document. Another possibility is that the tagged document represents a regular document that is marked with attached text tags, or labels.

3.7.3 Procedure

After the pre-task procedure had been completed, the participant was handed the anchor point scenario and a countdown for 300 seconds was set. By the end of the countdown, the participant was asked to rate the scenario using the SAM. Upon completion, the scenario and the participant's drawn diagram was retrieved, and placed together, out of sight of the participant. A countdown for 120 seconds was set and the participant was asked if everything was clear. At the end of the countdown or after all questions had been answering, whichever took the longest, the participant was handed their first scenario. The procedure for the scenarios was the same as for the anchor point, except for the fact that the researchers did not ask if the participant had any questions.

3.8 Post-task Interview

The third and final part of the session aimed to contextualise the SAM ratings, by investigating the participants' professional profiles and exploring the topic in their day-to-day work. The profile data were collected through a short questionnaire and were used to answer RQ2. Data concerning the participants' day-to-day work were obtained through a semi-structured interview in order to answer RQ3, but also

to triangulate the data from the laboratory experiment and to identify potential validity threats to the experiment.

3.8.1 Materials

The post-task interview has the least amount of materials, with no more than two pieces: the post-task manuscript and the participant profiling questionnaire. Nevertheless, those pieces are important and are closely related to the research questions, as well as the validation of the study.

3.8.1.1 Post-task Manuscript

Similar to the pre-task instructions, the post-task interview contained essential information that should not be excluded. In particular, none of the questions of the interview should be forgotten. Because of this, we composed a post-task manuscript to serve as a memory aid. The text is available in its entirety in Appendix G and a laminated copy was used during the sessions.

Conceptually, the manuscript was composed of three parts. The first served to introduce the participant to the questionnaire (see Section 3.8.1.2), and the second aimed, likewise, to convey the procedure for the semi-structured interview. The final part contained the common questions of this interview, which are repeated in Table 3.6. Each interview question (IQ) had a clear goal to either answer the research questions or assess the validity of the study. IQ1.1 and IQ1.3 are exceptions to this, as their aim is to set up questions IQ1.2 and IQ1.4, respectively.

3.8.1.2 Participant Profiling Questionnaire

The participant profiling questionnaire aimed to chart the participants' professional experience with software. The questions are repeated in Table 3.7. Each question (Q) had a clear goal to either answer the research questions or assess the validity of the study. A copy of the questionnaire is available in Appendix H.

3.8.2 Procedure

When the participant had rated their fifth and last scenario, the researchers read the post-task manuscript to the participant in the same fashion as the pre-task manuscript. As part of the post-task manuscript, the participant was handed the profile questionnaire and asked to fill it out.

The researchers retrieved the questionnaire upon completion and resumed the post-task manuscript. Should the participant choose to decline the use of audio recording, the researchers should acknowledge their choice and honour the decision. Otherwise, the researchers recorded the remainder of the session.

Most of the questions enumerated in the post-task manuscript are intended to be exploratory, open-ended questions. Because of this, the researchers are allowed to make ad-hoc changes. This includes rephrasing the question, asking new or follow-up questions, or skipping questions.

Table 3.6: The common questions of the semi-structured interview and what the purpose of each of them was.

ID	Description	Type	Goal
IQ1.1	Could you please tell us more about your daily work. What type of tasks do you normally encounter?	Open	Set up IQ1.2
IQ1.2	How do those tasks make you feel?	Open	RQ3.1
IQ1.3	Do you face challenges in those tasks?	Closed	Set up IQ1.4
IQ1.4	How do those challenges make you feel?	Open	RQ3.1
IQ1.5	Are those feelings frequent?	Closed	RQ3.2
IQ2	In contrast to challenging tasks, what sorts of feelings would you say you get from routine tasks?	Open	RQ3.1, RQ3.2
IQ3	Do you think that anything outside of this experiment did impact your responses today?	Closed	Validity
IQ4.1	Would you please tell us about how you experienced the code examples?	Open	Validity
IQ4.2	What about the software design in the examples?	Open	Validity
IQ5	What would you say are the differences between the scenarios we provided and software one encounters in industry?	Open	Validity
IQ6	Did you find SAM difficult to use or understand?	Closed	Validity
IQ7	That was all of the questions that we had for you. Is there anything you would like to add?	Open	Validity

Table 3.7: The questionnaire questions and what the purpose of each of them was.

ID	Description	Type	Goal
Q1	My highest level of completed academic education is	Closed	RQ2.1
Q2	My education major (e.g. computer science, electrical engineering, software engineering) was _____	Closed	RQ2.1
Q3	I have working experience with software for _____ years.	Closed	RQ2.2
Q4	My current role (e.g. architect, developer, tester, ...) is _____	Closed	RQ2.3
Q5	The programming language I am most experienced in is _____	Closed	RQ2.3, Validity
Q6	My currently preferred programming language is _____	Closed	Validity
Q7	Most of my working experience comes from the following domain (e.g. telecom, healthcare, finance, ...) _____	Closed	RQ2.3
Q8	Do you have any additional comments concerning this questionnaire?	Open	Validation

When the manuscript was completed and the researchers had no further questions, the session may be terminated, even if the 90 minutes are not used up. If the session lasts longer than 90 minutes, it should be terminated as soon as possible, regardless of what stage the session is in.

3.9 Variables

In contrast to typical software engineering research, research in psychology often attempts to measure small effects in a noisy environment [43]. As opposed to investigations of technical phenomena, empirical research with human participants is inherently complex and highly variable [55]. As noted in Section 2.2.2, concerns about traditional statistical approaches have been voiced in the field psychology and one of the suggested remedies is using Bayesian statistics (see Section 2.4). We have decided to adhere to this advice. In Table 3.8, we list the variables, together with their source of data, scale, and range, that were investigated in the study and will be used to create the statistical model.

The independent variables are the components of the dimensional framework (see Section 2.2.1.1) of affects are valence (V), arousal (A), and dominance (D). They are all measured with the SAM (see Section 2.2.1.2) and will be predicted by the remaining variables.

The example, which identifies the software scenario and its version, and the number of logical entities (ENTITIES) are independent variables that model the characteristics of the code examples that were used in the experiment. Of these, only

Table 3.8: The main factors that will constitute the statistical model.

Name	Variable Type	Source	Scale	Range
V	Dependent	SAM	Ordinal	1–9
A	Dependent	SAM	Ordinal	1–9
D	Dependent	SAM	Ordinal	1–9
EXAMPLE	Independent	Design	Nominal	ScA-H–ScE-L
ENTITIES	Independent	Design	Ratio	2–13
ID	Independent	Sample	Nominal	1–40
EDU	Independent	Questionnaire	Ordinal	-
MAJOR	Independent	Questionnaire	Nominal	-
EXP	Independent	Questionnaire	Ratio	-
ROLE	Independent	Questionnaire	Nominal	-
LANG	Independent	Questionnaire	Nominal	-

LEVEL is related to any of the research questions, namely RQ1.1. The others are concerned with the validity of the experiment.

The remaining variables model the traits of the sample. Those were the participants (ID) themselves and their professional characteristics (RQ2). The participants' level of education (EDU) and major aims to answer RQ2.1, while the purpose of work experience (EXP) is to answer RQ2.2. Trying to answer RQ2.3 is the goal of their role and the language they are most experienced in (LANG).

3.10 Analysis Procedure

The research objective of this thesis was to investigate the relationship between TD and affective state from the point of view of software practitioners. In order to achieve this objective, we have chosen to apply a mixed-methods approach by combining a quantitative method with a qualitative one. The quantitative method was a laboratory experiment and will be analysed with Bayesian statistics, while the qualitative method was semi-structured interviews that are understood through a thematic analysis.

3.10.1 Quantitative Analysis

In Section 2.2.2, we highlighted that a central issue in the field of psychology is the replicability of results, for which many researchers have proposed methodological remedies. One of the commonly voiced concern is the use of p-values and a transition to Bayesian statistics (see Section 2.4) has been suggested in order to combat the problem.

We have followed the advice to use Bayesian statistics and will analyse our data in the statistical software *R* [58] through the use of the *rethinking* package [59] for Bayesian statistics. The definitions of our statistical model is presented as source code below. Note that we have only included the code for the valence dimension of

affects, as the other two are analogous.

```
dat <- list(
  V = d$V,
  A = d$A,
  D = d$D,
  EDU = d$EDU,
  alpha_edu = rep(2,6),
  MAJOR_index = as.integer( d$MAJOR ),
  ROLE_index = as.integer(d$ROLE),
  EXAMPLE_index = as.integer(d$EXAMPLE),
  LANG_index = as.integer(d$LANG),
  ID_index = as.integer(d$ID),
  EXP = d$EXP,
  ENTITIES = d$ENTITIES
)

model_V <- ulam(
  alist(
    V ~ ordered_logistic( phi , kappa ),
    phi <- id[ID_index] + example[EXAMPLE_index] + bEntities*ENTITIES
    + bEdu*sum( delta_edu_j[1:EDU]) + major[MAJOR_index] + bExp*EXP
    + role[ROLE_index] + lang[LANG_index],
    kappa ~ normal( 0 , 1.5 ),
    c(bEntities, bEdu, bExp) ~ normal( 0 , 1 ),
    id[ID_index] ~ normal(0, 1),
    example[EXAMPLE_index] ~ normal(0, 1),
    major[MAJOR_index] ~ normal(0, 1),
    role[ROLE_index] ~ normal(0, 1),
    lang[LANG_index] ~ normal(0, 1),
    vector[7]: delta_edu_j <<- append_row( 0 , delta_edu ),
    simplex[6]: delta_edu ~ dirichlet( alpha_edu )
  ), data=dat , chains=3 , cores=3 )
```

All priors are weakly informative [45], since the evidence found in previous research on the topic of ATD and affects was deemed insufficient for incorporating into the model. Consequently, the model will make more conservative estimates than is typical for this kind of analysis.

3.10.2 Qualitative Analysis

To gain a broader understanding of the topic of this thesis, the quantitative method was complimented with a qualitative analysis. This analysis had the specific goal of answering two of our research questions, RQ3.1 and RQ3.2, which sought to explore how aware software practitioners are of any potential affects caused by software engineering tasks and, respectively, the frequency of such affects. Hence, the analysis

was explicitly driven by an analytic interest in the area, for which a deductive thematic analysis is suitable [53].

Because the research on the topic is limited (see Section 2.3), we deemed it more prudent to conduct a surface level analysis of the interviews. The risk of overinterpreting the answers, and bias the results to our own personal beliefs, would be substantial without the regulating effects of independent findings. Consequently, we opted for performing the thematic analysis on the explicit level.

Finally, the data was inspected from a realist perspective in order to avoid ethical concerns. The sample size equalled 40 and was therefore considered too small to investigate socio-cultural contexts without compromising confidentiality. The distribution of demographic characteristics was speculated to be quite skewed, which might lead to participants with distinguishing backgrounds to be identifiable, which we consider unacceptable for this thesis.

With these characteristics established, the thematic analysis was carried out, in accordance with the guidelines of Braun and Clarke [53], as follows. First, the interviews were transcribed, verbatim, from the audio recordings. Second, potential themes were outlined by collating and the texts were coded according to the themes. Next, the coded parts of the interview were compiled and synthesised into new themes. The new themes were used in a second coding of the transcribed interviews in order to verify that the synthesised themes were representative of the contents of the interviews. Any themes that were deemed insufficiently representative were discarded. The remaining themes were presented in a thematic map.

4
Results

The main objective of this thesis was to investigate the relationship between TD and affective state from the point of view of software practitioners. To achieve this objective, a mixed-methods approach was selected to answer three main research questions.

Because the study of affects is somewhat infrequent in software engineering research, we will recap what the affective dimensions (see Section 2.2.1.1) of the valence, arousal, and dominance describe. Valence expresses how much attraction one finds to a stimulus. Arousal corresponds to how mentally awake and reactive one feels in reaction to a stimulus. Dominance represents how challenging a task is a perceived in relation to one's skills.

This section will report the findings of this study and is structured in the following way. First, an account for deviations from the planned procedure is given. Second, an overview of the collected data is presented. Finally, the results for each of the three main research questions are detailed.

As we have previously highlighted (see Section 2.2.2), the field of psychology is experiencing a replication crisis, in part due to the reliance of significance testing. When the phenomena under study are small and highly variable, and the context noisy, that approach is problematic [43]. In spite of this, we have chosen to phrase our results in those terms for two reasons. First, the use of p-values and significance testing is still convention and by adopting equivalent terminology, we believe that the results will be accessible to a wider audience. Second, interpreting the data without clear cut thresholds often requires lengthy argumentation that is atypical for the results section. Instead, we defer such discussions to Section 5, where we will reinterpret the data.

4.1 Method Deviations

During the data collection and analysis phases, a couple of situations surfaced that were unaccounted for. Those will be presented in this section, together with a short description of how they were managed.

4.1.1 Data Collection Deviations

Most of the method deviations occurred during the sessions and had to be resolved quickly or risk introducing additional bias. Because of this, the risks could not be adequately analysed before reaching a decision. In addition, when a decision had

been made, it was incorporated in the remaining sessions in an attempt to keep the experiment as consistent as possible.

The first situation was concerned with software modelling. During the pilot studies, all participants readily constructed diagrams of the scenarios. This was not the case in the proper study, where several participants expressed disinterest in modelling or were unfamiliar with the practice.

Because the experiment was concerned with measuring affects, this situation presented a dilemma where both alternatives introduced threats to validity. On the one hand, the procedure could be adapted, by stating that modelling was optional. This would make those participants more comfortable with the situation and, therefore, mitigate the risk of introducing another source of stimuli that would influence their ratings. On the other hand, this would mean that the experiment would be somewhat inconsistent, as the participants, in some sense, would not perform the same tasks. We decided to make the modelling optional.

The second situation was concerned with the deacclimatisation periods. During the pilot studies, the participants accepted the silence as part of the experiment. In the main study, several participants initiated conversations with the researchers. Similar to the situation with software modelling, the procedure was adapted to allow this situation as the alternative was deemed too threatening to the validity of the study.

In addition to these more overarching issues, there were a few situations that occurred in individual sessions. While they all are comparatively minor and therefore should at most have limited impact on the validity of the study, they are listed for transparency.

One session was held in Swedish, because the participant was not comfortable with having an entire conversation in English. In one session, the participant chose to fill out the SAM before the time was due, in spite of repeated clarifications. One participant received the scenarios in a different pattern (HLHLH) than was intended. Finally, one participant chose to falsely report their level of academic education in the questionnaire. Despite the participant verbalising that they had not completed their master studies and the researchers pointing out that there was an alternative for some master studies, the participant filled out having a master degree.

4.1.2 Analysis Deviations

Although the plan was to include all 40 interviews in the thematic analysis, the questions under investigation were quickly saturated, and also corroborated by [52]. Consequently, 10 representative interviews were identified and selected for the analysis.

This assessment was made in the first phase of the thematic analysis (see Section 2.5), during which the researchers familiarise themselves with the data. Both authors attended all the interviews and continuously discussed the data among themselves, and agreed that the answers to the research questions were, to a high degree, unanimous. As a result, the analysis would become saturated with a comparatively small data set. This meant that analysing more data would quickly lead to diminishing returns and thus provide marginal benefit for the purpose of this thesis.

This saturation does not necessarily mean that software practitioners are in agreement on the topic of affects, as the situation can be explained by several causes. Perhaps most probable is the proposition that the research questions were narrow in scope. Because of this, it is likely that other research questions and other types of thematic analyses would not have encountered consensus.

4.2 Data Sets Description

As explained in the methods section, this study collected data, from 40 participants from 12 companies, through three different strategies. First, each participant self-reported their affective state in response to ATD through a laboratory experiment administrating the SAM. Second, every participant self-reported their professional background with a questionnaire. Third, each participant answered questions in a semi-structured interview, concerning how they perceived the study and their perception of code maintainability and feelings in general.

This structure is mirrored below, where we present the collected data in the aggregate. The intention is to acquaint the reader with the data on a high level of abstraction, since the data sets are quite rich and multifaceted. Rather than attempting to answer the research questions, this section will outline the data and highlight its more general characteristics.

At the end of this section, diagnostic findings regarding the data are reported. The complete data set that was used in the statistical model (see Section 3.10.1) can be found in Appendix I. Those data have been processed according to the descriptions in this section, but the unprocessed raw data are available upon request in exchange for an appropriate assurance of maintaining participant confidentiality.

4.2.1 The SAM Response Data Set

The data set obtained from the measurement sitting consists of participants' self-assessed ratings of their affective state. Each rating constitutes one score on each of the three dimensions of the PAD framework: valence, arousal and dominance (see Section 2.2.1.1). The scores were obtained through the use of the SAM and ranges from one to nine on an ordinal scale.

The data was collected during a laboratory experiment where the 40 participants reported their affects in response to five different software scenarios, yielding a total of 200 ratings. These ratings are presented by respective dimension in Figures 4.1-4.3. The figures show the histogram of each respective affective dimension, as well as the corresponding cumulative proportion diagram. In each of the figures, we can see that the entire range of the instrument was covered by the scenarios.

Every software scenario existed in two functionally equivalent versions, but which had different amounts of ATD. The version with the higher (H) amount embodied a unique software design smell, which had been refactored in the version with lower (L) amount of ATD. While every participant received each scenario, which version they were exposed to was determined with equal and independent probability.

Figure 4.4 presents a histogram of how many participants rated each version of each scenario. It is clear that no version deviated far from the mean value of 20, as

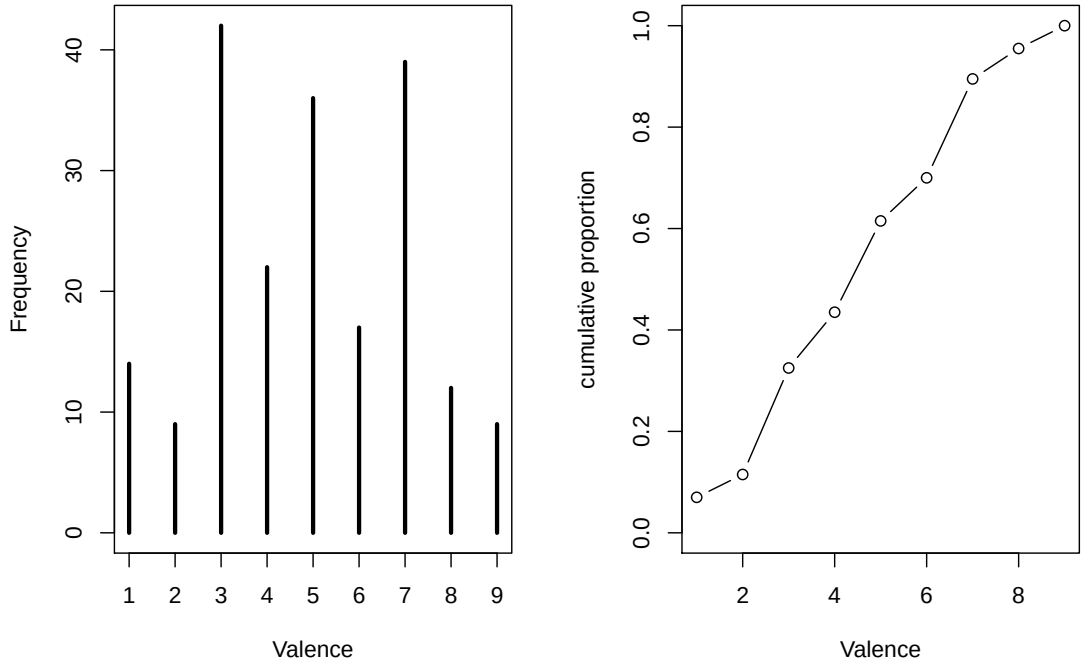


Figure 4.1: The distribution of responses for valence.

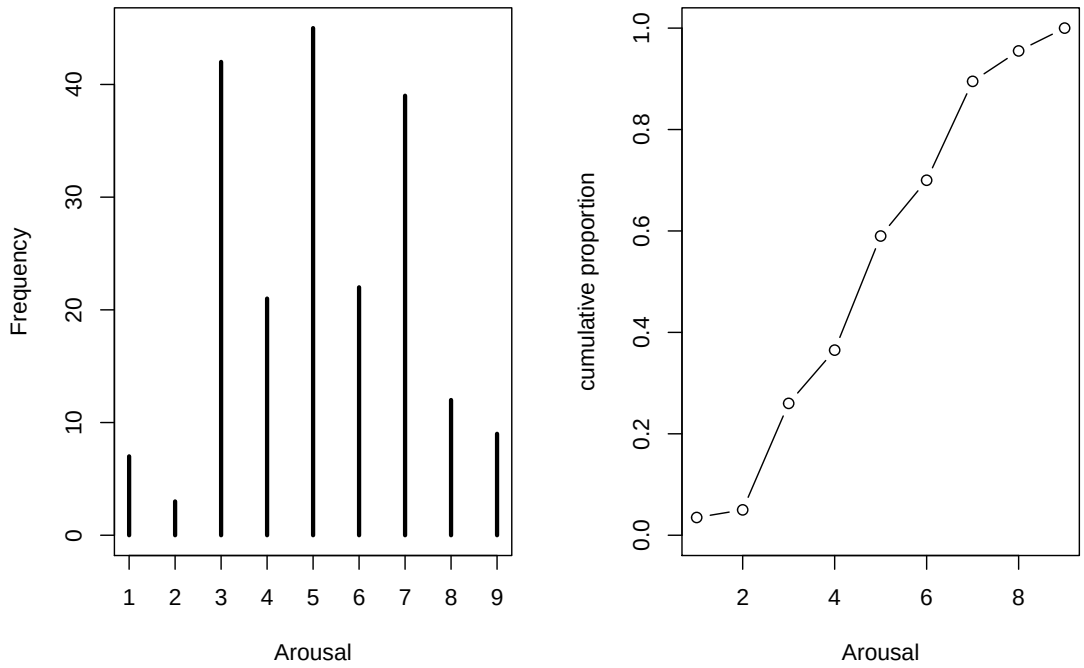


Figure 4.2: The distribution of responses for arousal.

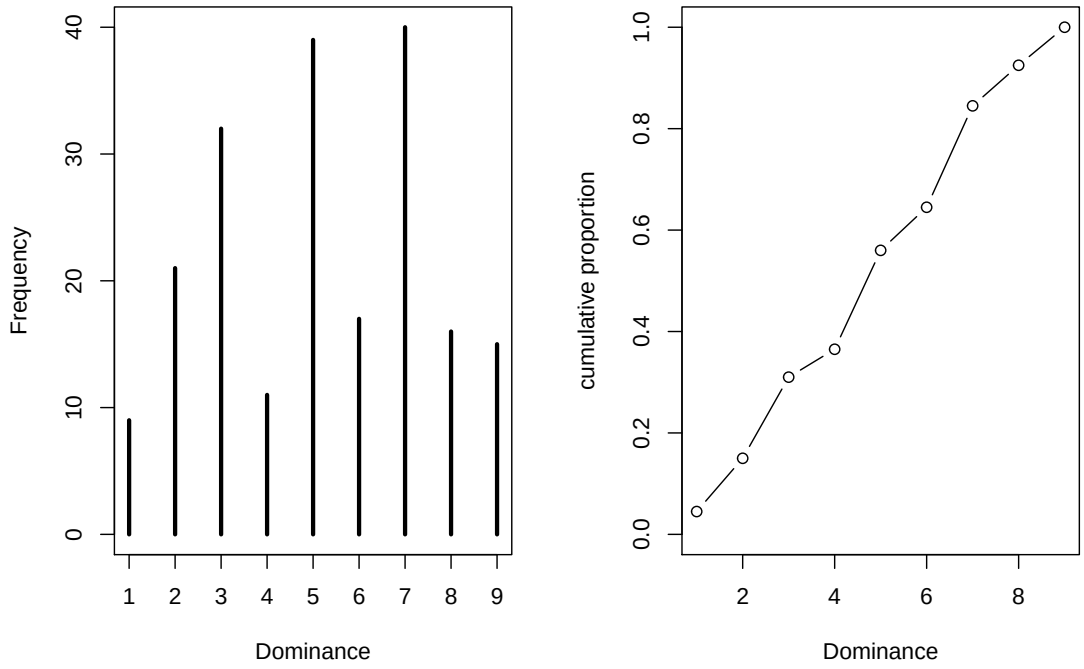


Figure 4.3: The distribution of responses for dominance.

each one was rated by at least 15 unique individuals. This indicates that no version was underrepresented nor overrepresented.

In addition to each scenario existing in one high and one low version, the scenarios themselves were different. As illustrated in Figures 4.5-4.7, this was reflected in the participant's ratings of them. For different scenarios, the distribution of the ratings varies for all of the three dimensions. The tendencies are both stronger and more certain for the dimensions of valence (see Figure 4.5) and dominance (see Figure 4.7) than for arousal (see Figure 4.6), but there is still too much uncertainty at the 95 % HDI to make any conclusive statements about the effect of the scenarios. However, this could be interpreted as an indication that specific scenarios could influence the ratings of the participants.

4.2.2 The Profile Questionnaire Data Set

In order to obtain data about the professional background of the participants, the participants were asked to fill out a profiling questionnaire. These data concern some general professional characteristics of the sample and are analysed in Section 4.4 to understand if and how individual differences influence how software smells are perceived. The questionnaire comprised 8 questions (see Section 3.8.1.2), the responses for which are presented below, in order.

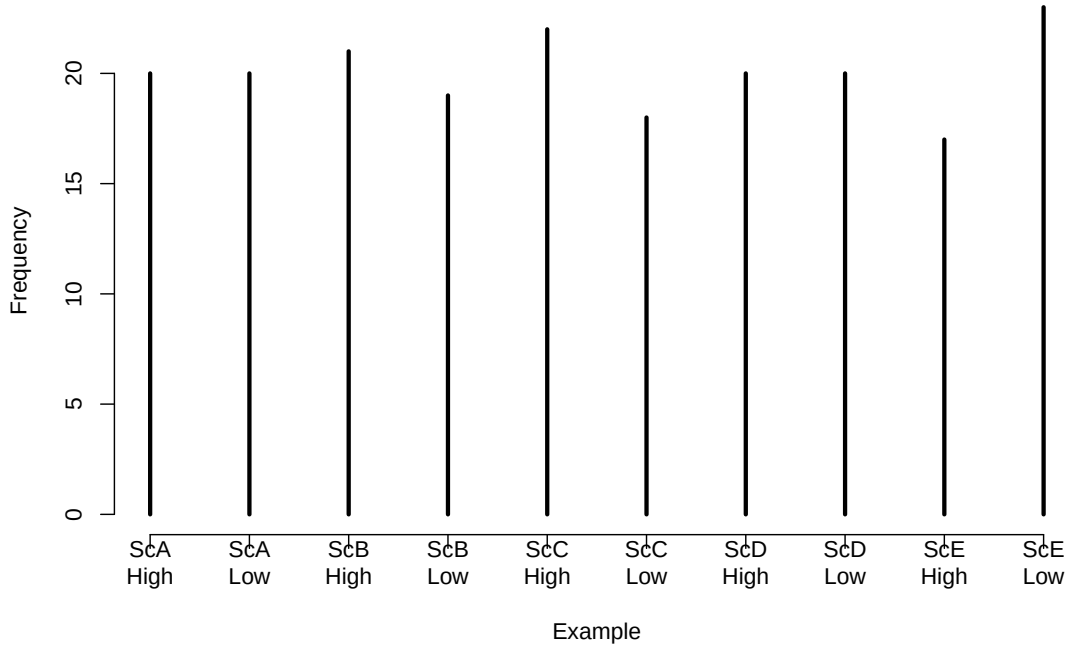


Figure 4.4: The distribution of the occurrence of the five scenarios.

4.2.2.1 Highest Levels of Completed Academic Education (Q1)

The first question of the questionnaire concerned the participants' level of formal education and offered seven response alternative, in addition to the option of filling out other alternatives. Three of the participants chose that option and specified either "KY" or "magister". The former is an abbreviation of "Kvalificerad Yrkesutbildning", which can be translated into two years of cooperative education. The latter can be translated to a one-year master degree.

These responses present a minor dilemma. On the one hand, introducing them as new categories of educational levels would change the data type from ordinal to nominal, because there is no inherent order among, for example, a one-year master degree and some master studies. As a result, the expressiveness of the remaining 92.5 % responses would be compromised.

On the other hand, if the responses were to be incorporated into the existing categories, this might be misrepresentative. In the event that there are important differences between the categories, the data might be fallacious. In this instance, we judged the latter approach as less detrimental and decided to incorporate KY into some bachelor studies and magister into some master studies.

Figure 4.8 presents the resulting data, from which can be read that the clear majority of participants had completed their education at either the bachelor or master level. The remaining options were comparatively scarcely selected and the sample did not include anyone who was part-way through their Ph. D.

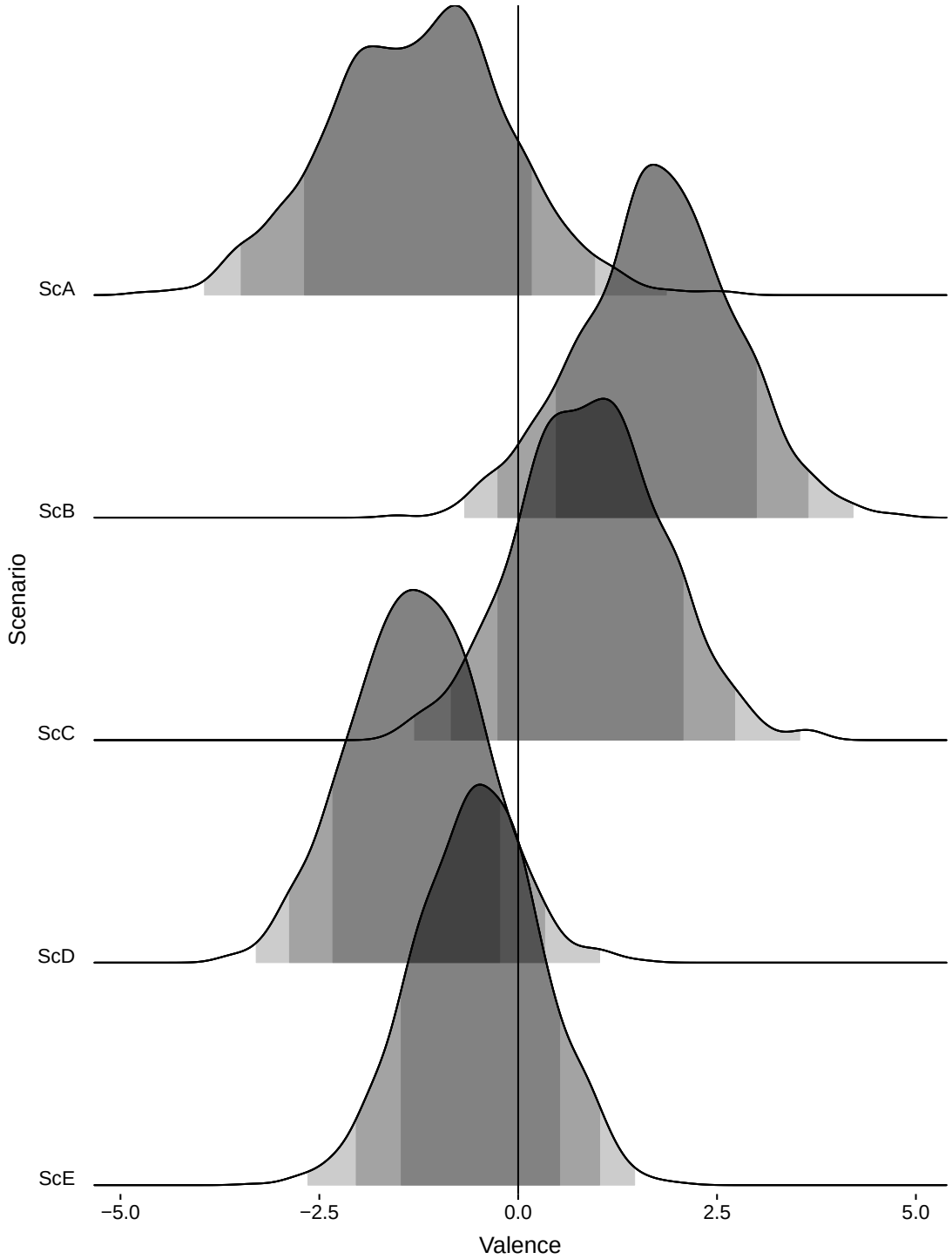


Figure 4.5: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effects of the different scenarios on valence.

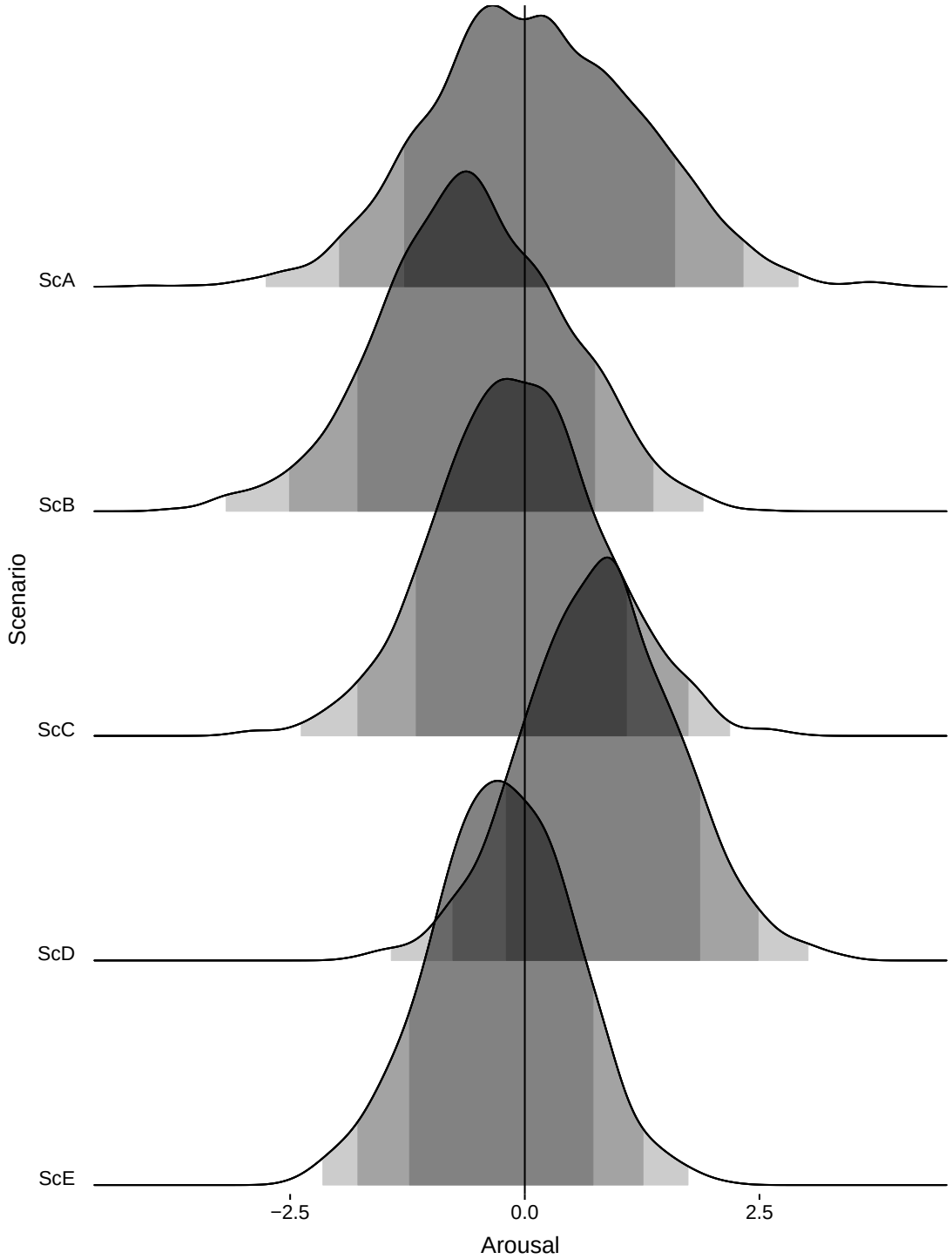


Figure 4.6: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effects of the different scenarios on arousal.

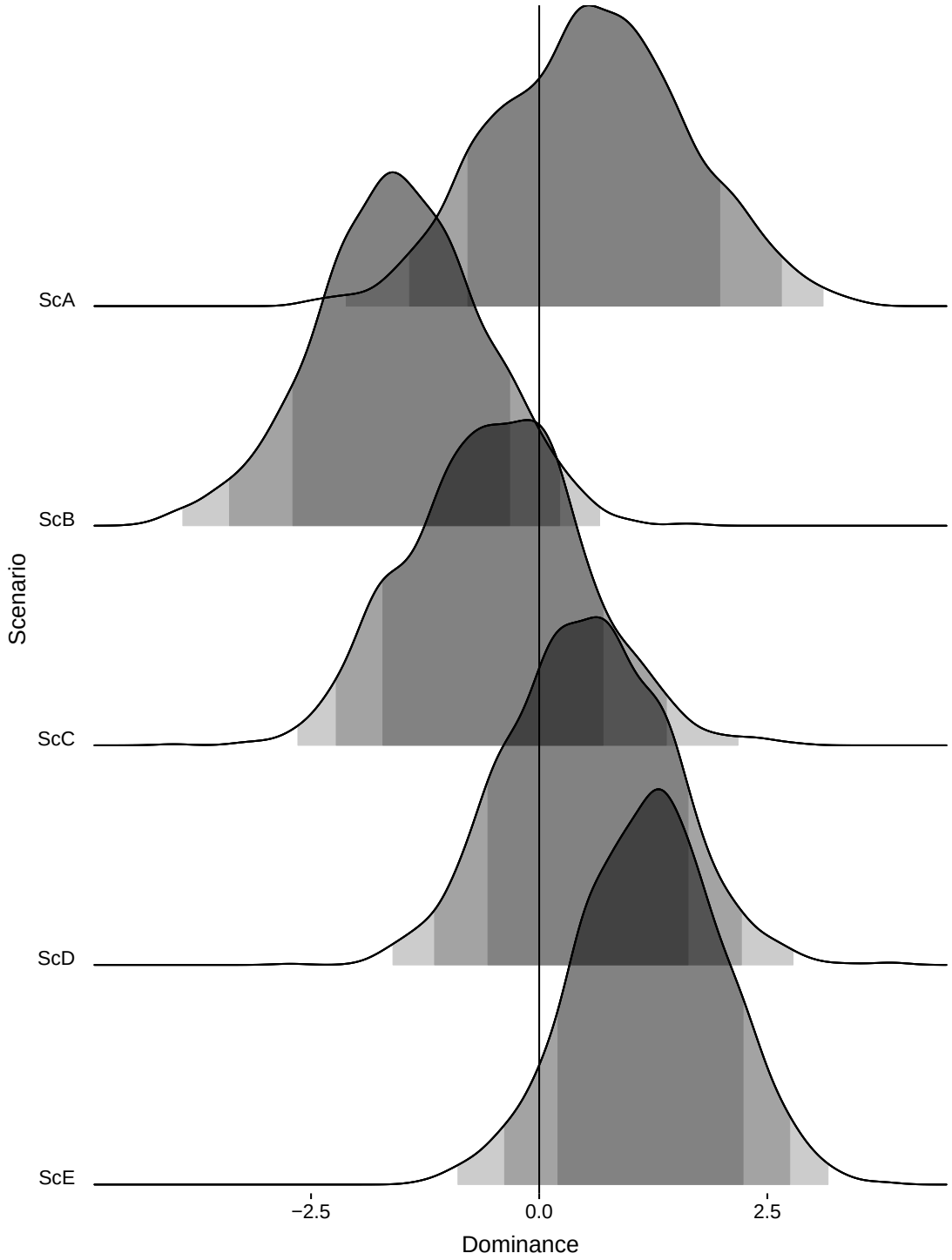


Figure 4.7: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effects of the different scenarios on dominance.

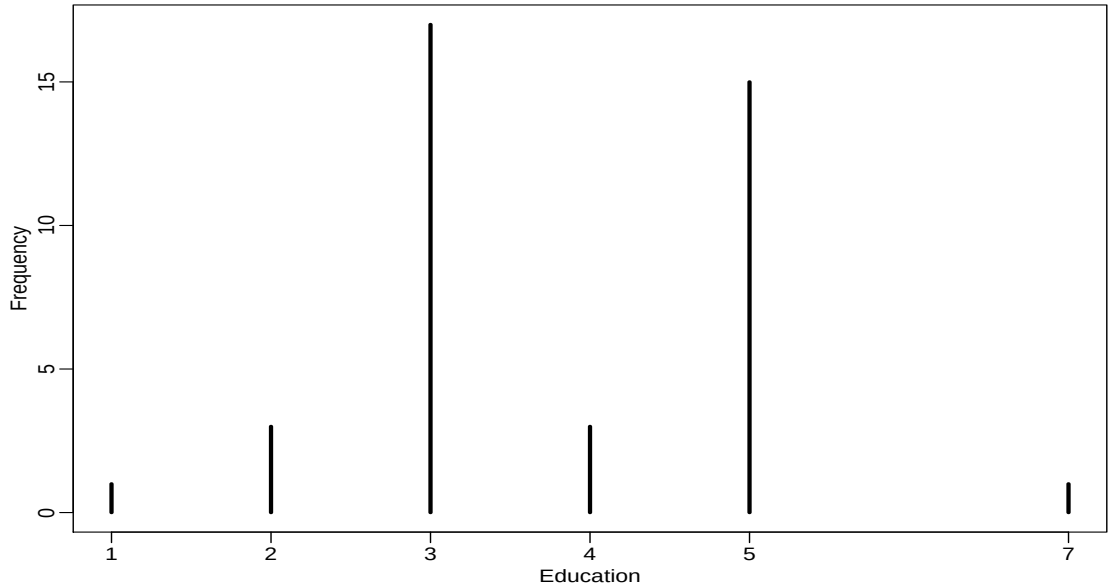


Figure 4.8: The distribution of the participants' highest level of completed academic education. The data type is ordinal and the levels, in order, correspond to 1) None; 2) Some bachelor studies; 3) Bachelor degree; 4) Some master studies; 5) Master degree; 6) Some Ph. D. studies; and 7) Ph. D.

4.2.2.2 Education Majors (Q2)

The answers to the question of what the participants had majored in were quite diverse, with a total of sixteen unique responses. As shown in Table 4.1, many of the categories were quite specific and could be considered subfields to more general subjects. Additionally, each of the specific majors typically had no more than one respondent each.

For these reasons, it was decided to incorporate the subfields into the more general subjects. A histogram of the resulting data is shown in Figure 4.9. Software engineering and computer science were the most common majors and each of the remaining ones had only a few respondents.

4.2.2.3 Work Experience with Software (Q3)

The third question of the questionnaire addressed how much work experience, in years, the participants had with software. Four respondents included half-years in their answer, which is a level of granularity that this study is not concerned with. Consequently, all responses were rounded down to the closest integer. Specifically, the affected responses were two instances of 1.5, one of 2.5 and one of 4.5 years.

The resulting data is illustrated in Figure 4.10. Less experienced practitioners are more common in the data set; more than half of the participants had less than ten years of experience. Note also the large experience gap between the most experienced and the second most experienced participants.

Table 4.1: The frequencies of the participants' self-reported majors and how those were translated to more general fields of study.

Major	Frequency	Superfield
Computer Science	13	Computer Science
Software Engineering	9	Software Engineering
N/A	3	None
Informatics	3	Software Engineering
Electrical Engineering	2	Electrical Engineering
Computer Applications	1	Software Engineering
Digital Forensics	1	Computer Science
Electrical Engineering and biomedical engineering	1	Electrical Engineering
Engineering Physics	1	Engineering Physics
Game and Software Engineering	1	Software Engineering
IT marketing in businesses	1	Business Administration
Security and mobility	1	Computer Science
Software Engineering, Computer Science	1	Software Engineering
System Architect	1	Software Engineering
System science	1	Software Engineering

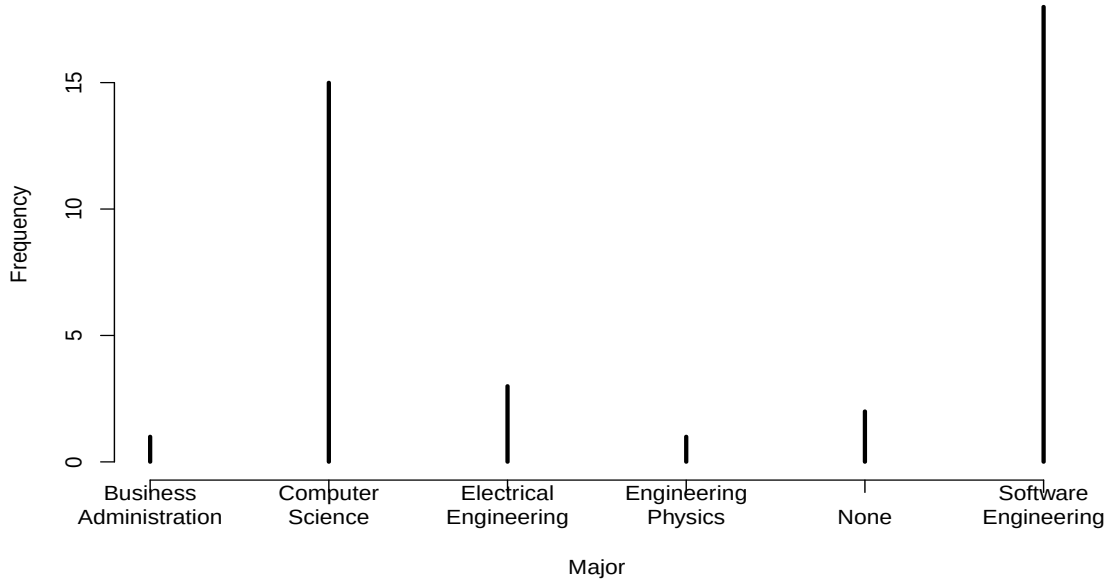


Figure 4.9: The distribution of the participants' majors, after aggregating them into more general subjects.

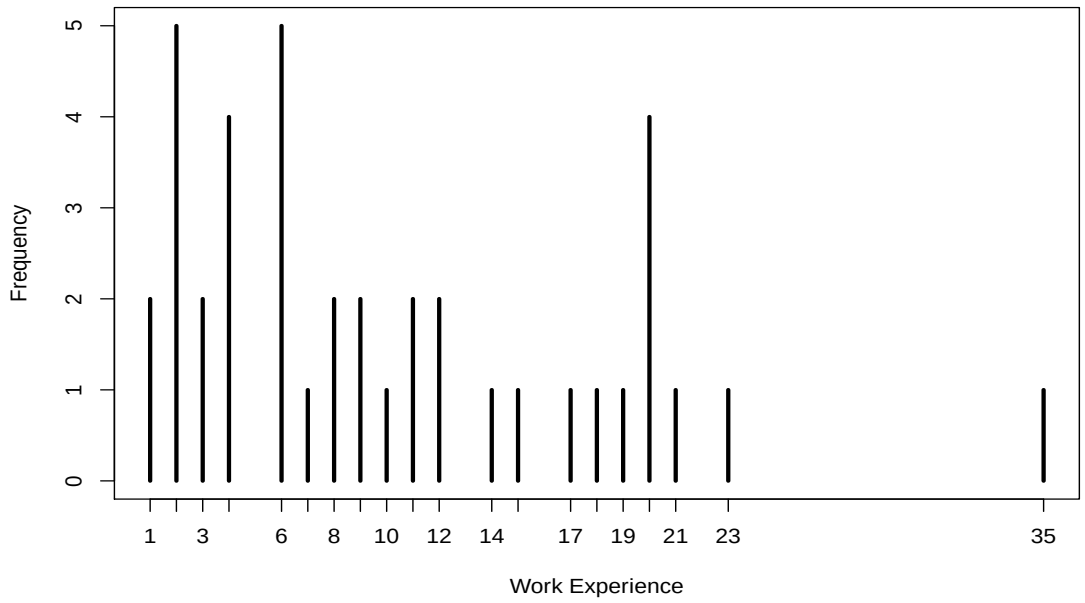


Figure 4.10: The distribution of the participants' work experience with software in years.

4.2.2.4 Work Roles (Q4)

The sample contained ten different work roles, which are listed in Table 4.2, together with the frequency of each. Note that summing the frequencies yields a higher number than the sample size. The explanation is that some respondents reported having multiple roles.

Because of this discrepancy, the responses were reduced to a single role. In most cases, the role selected was simply the one first listed. However, in cases were one of the roles was an architectural role, that one was chosen as the study is concerned with architectural TD. Similar to the majors, some of the work roles were incorporated into more general roles.

As can be seen in Figure 4.11, the majority of the respondents were developers, even though architects were frequent participants.

4.2.2.5 Programming Languages Most Experienced In (Q5)

Question five was concerned with what programming language the participants had the most experience in. As can be seen in Figure 4.12, half of the participants responded with C# and about a quarter answered Java. The remaining responses were spread out in ones or twos among eight other languages.

4.2.2.6 Preferred Programming Languages (Q6)

Figure 4.13 provides an overview of the participants' preferred languages. When compared with Figure 4.12, it is clear that many characteristics are shared between

Table 4.2: The frequencies of the participants' self-reported work roles and how those were translated into more general ones.

Role	Frequency	Encompassing role
Developer	29	Developer
Architect	8	Architect
Senior developer	3	Senior developer
Scrum master	2	Manager
Software Engineer	2	Developer
Tester	2	Tester
Data scientist	1	Senior developer
Engineering Manager	1	Manager
Software Developer	1	Developer
Team Lead	1	Manager
	50	

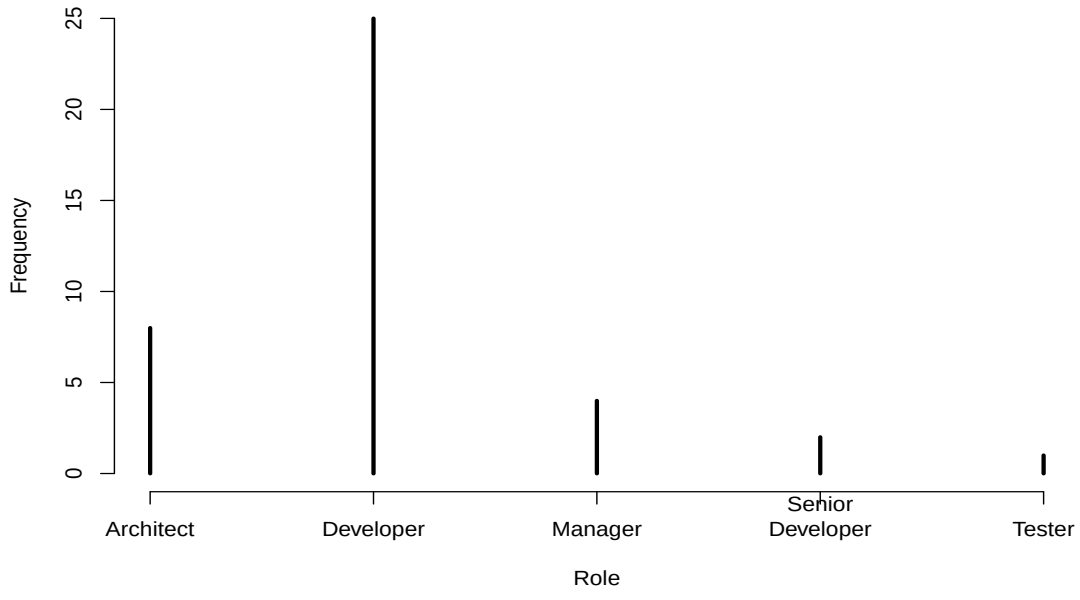


Figure 4.11: The distribution of the participants' work roles, after aggregating them into more general ones and reducing each response to a single role.

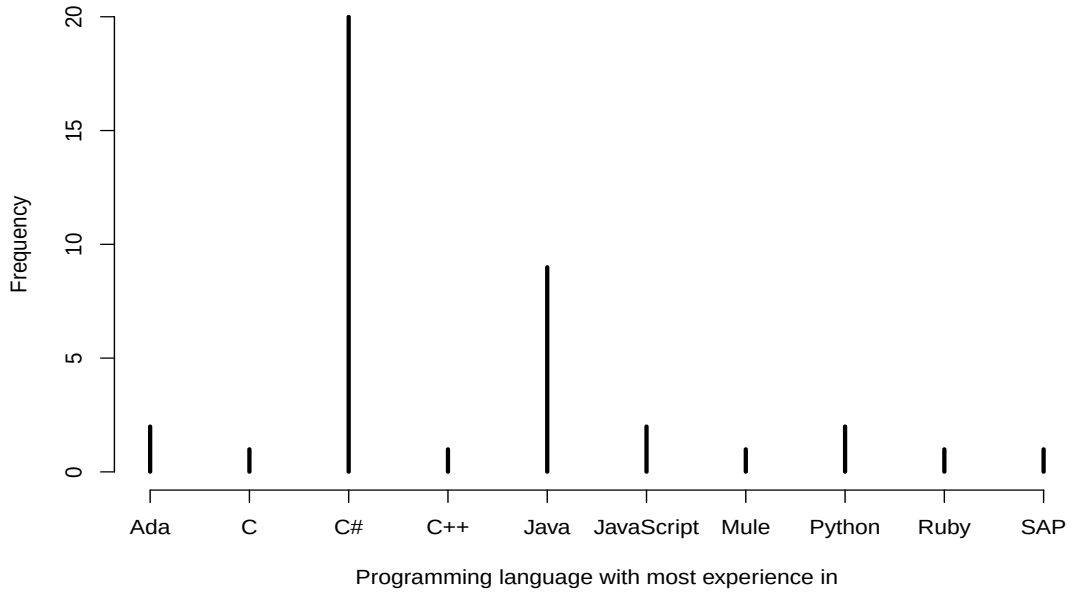


Figure 4.12: The distribution of what programming languages the participants were most experienced in.

them. Here, as well, half the participants respond with C# and about one fourth with Java, while the remaining languages get a few responses each.

Indeed, the full data set reveals that the majority of respondents prefer the language they have the most experience in, while nine participants do not. More specifically, one participant prefers Go to Python, another Java to SAP ABAP, one C# to JavaScript, one JavaScript to C#, one Python to C#, one Java to Mule, one C# to Java, one Go to C, and two TypeScript to Java.

4.2.2.7 Domains Most Experienced In (Q7)

The last question of the questionnaire was concerned with what domains the participants were most experienced in. The data presented in Table 4.3 demonstrate that the sample covered a diverse set of domains and that the most common domains were logistics, finance, and telecom. Note that the frequency is higher than the number of participants due to some participants answered having equal work experience in multiple domains.

4.2.2.8 Additional Comments (Q8)

Four participants left comments that addressed validity threats, all of which were accounted for in the design of the study. Two of the comments were concerned with the participant's experience, or lack thereof, with the Java programming language, which the scenarios were written in. These were mitigated by the structure of the statistical model, which used programming language as a predictor.

Table 4.3: The frequencies of the participants' self-reported domain of most work experience.

Domain	Frequency
Logistics	7
Finance	7
Telecom	5
Telematics	3
Automotive	2
Defence	2
Education	2
Pensions	2
Information Technology	2
Insurance	2
Transportation	2
N/A	2
Administrative Software	1
Analytics	1
Aviation	1
Balanced Scorecard	1
Business Administration	1
Business to Business	1
Connected Vehicles	1
Enterprise Resource Planning Systems	1
Government	1
Linux Backend	1
Maps	1
Planning	1
Productivity tools	1
Retail	1
Skin care / beauty	1
Software as a Service	1
Taxes	1
Web	1
Welfare	1
	57

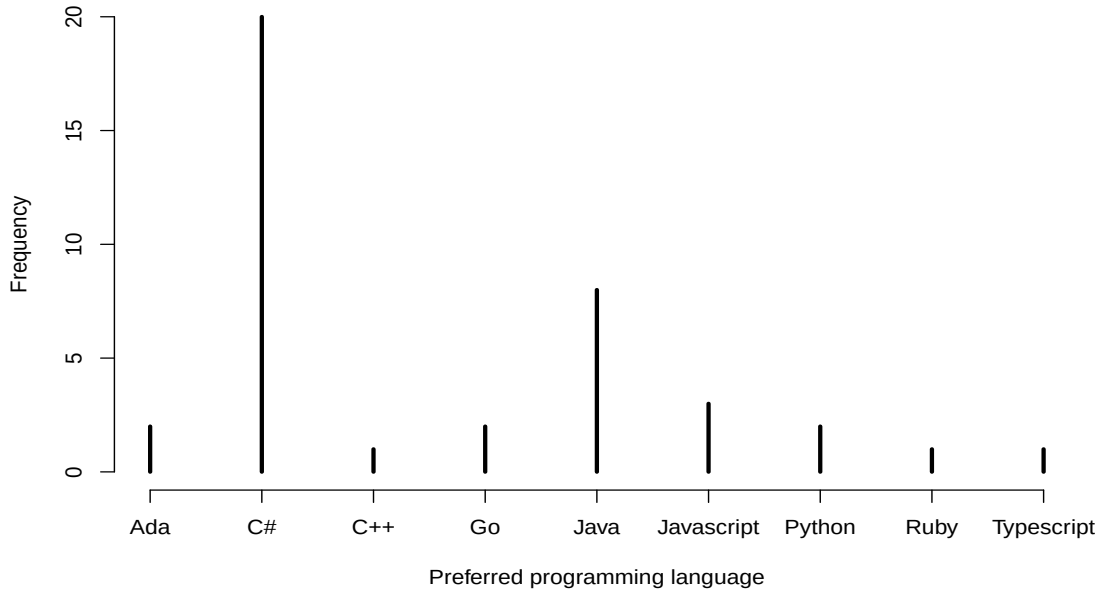


Figure 4.13: The distribution of what programming languages the participants preferred.

One comment pointed out that the SAM ratings could depend on the participant's mood and thus vary from day to day. Further, two comments expressed that the participant enjoyed attending the session. These kinds of effects were mitigated by the within-subject design of the experiment.

In addition, one comment suggested that the duration of the different scenarios should vary as the experiment progressed.

4.2.3 The Interview Data

The data set used for the qualitative analysis consisted of the digital recording of 10 interviews out of 39 performed and recorded interviews (one interviewee asked not to be recorded). These were selected, randomly, to be representative for all interviews, with regards to the research questions investigated in this study, but were at the same time also diverse in terms of professional background. In total, the duration of the interviews was 188 minutes, each individual interview lasting between 13 and 30 minutes.

This section presents the results of the thematic analysis, introducing the main themes found during the qualitative analysis. Then, each of the main themes is described in more depths, after which other findings not covered by the thematic analysis are presented. Note that, since RQ3 investigates potential changes in practitioners' affective states in general (see Section 3.2), the themes reported include peripheral areas, in addition to themes directly related to technical debt.

From the thematic analysis, three main themes were identified, as shown in the thematic map in Figure 4.14. In the map, the ovals represent groups of affective

states reported by the participants in relation to their work, and constitutes the main themes of the map. The rectangles are the stimuli described as causing the affects, forming the sub-themes of the map. The connecting lines show what themes were often discussed in conjunction with each other.

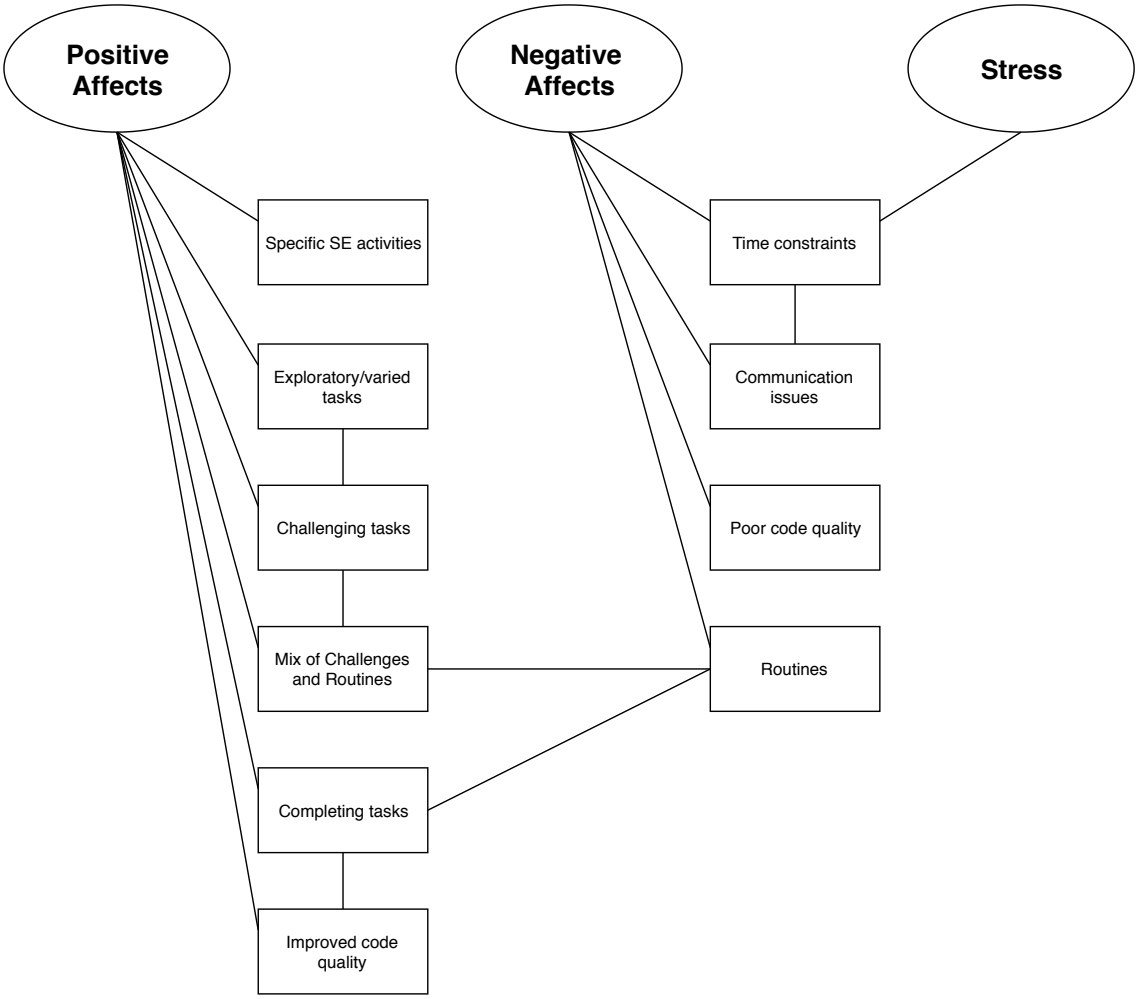


Figure 4.14: The map of the themes found in the thematic analysis.

The first main theme is positive affects, which concerns work situations that the participants reported preferring, liking, or appreciating. The participants mostly described the situations in terms of themselves as central actors, such as one participant saying *"once I have a clear idea of what I wanna achieve, then that feels encouraging. I like that. I like solving things"*.

In contrast, the second main theme, negative affects, describe work situations that the participants reported as causing frustration, anger, annoyance, or boredom. The participants typically describe these situations as though they had little or no agency in the situation. For example, positive affects were expressed as *"if I believe that I can make a difference and we can make something, then I like to really change things. So, I would like to investigate and then do the changes. Get things done"*, while negative one were *"sometimes you actually encounter some area that makes you really unhappy to be in."*

The last main theme was stress, which concerns work contexts in which the participants report experiencing different types of stressors, both negative and positive. For example, one participant talked about stress in somewhat detrimental terms, stating that "*things happening near deadline is always more stressful than when you have a lot of time to fix* [problems]". In contrast, another participant talked about some stress as a positive affect, saying *"I'm not the person that reacts really badly to stress, so I kinda like to have like a small amount of stress"*. Stress is represented in the thematic map as its own main theme due to there being no clear consensus on whether stress was a positive or negative affect among the participants.

4.2.3.1 Positive affects

Each of the main themes has one or more sub-themes that further specifies what the participants reported as causing that main theme, or affective state. As we can see, software practitioners receive positive affects from several sources. There are various specific software engineering activities that they enjoy, but the type of tasks and the outcome are also important.

Several participants talked in more broad terms of what they enjoy in software engineering tasks, rather than specific types of activities. The commonalities include a variation of tasks that do not constrain them too much, such as challenges with elements of exploration or learning. Examples of expressing positive affects from exploration include *"I find it very motivating to work with the, like, unknown outcome, because it's like pioneer work"*, as well as *"it's always looking for solutions and learning new things and figuring things out. To me it's a very good feeling"*.

Participants often expressed that they want some mix of challenging and routine tasks. However, too many challenges is often reported as tiring and, in the reverse case, too many routine tasks would be boring. The preferred proportions of the mixture seem to depend on the individual and vary over time. As one participant put it, while talking about routine tasks *"it's nice if you have had challenging tasks before. It's always the mix that makes up the good working environment, I think, because challenging tasks is fun, but you always have challenging tasks you get very tired and, yeah, at the end of the day, when you come home you're sleepy. So, routine tasks is good in some way, I think, but only routine tasks is, then you're bored"*.

We also observe that software practitioners like it when the outcome of their work improves the code quality, describing feelings of satisfaction. This is often discussed in conjunction with the completion of tasks. For example, one participant said *"I like solving things. Fixing things, getting stuff done. I think that's what drives me"*

4.2.3.2 Negative affects

In contrast, we see that routine tasks, poor code quality, communication issues, and time constraints are sources for negative affects. As noted above, these sources are commonly phrased in terms of the work environment, rather than the result of the participant's actions.

While it is evident that the participants think that they understand the organisational need for routine tasks to be solved, an abundance of such tasks causes

boredom, as mentioned above. As one participant put it: *"Sometimes I have this like repetitive tasks - 'Monkey Tasks', which I call them - they are never fun but has to be done"*.

Seeing and working with code of inadequate quality is often described as frustrating and it appears as though the existence of such code, as opposed to producing it, is attributed to such affects. One participant even stated that they get *"very annoyed"* when faced with bad code.

Several participants attributed communication issues to negative affects. Most further reported that communication issues are more frequent near time constraints, such as deadlines, where there is insufficient time to work out the issues. In addition, technical or physical limitations on communication, e.g. the use of chat systems, were described as problematic.

4.2.3.3 Stress

Finally, stress was a theme that was frequently reported, but the only consistent sub-theme was time constraints. In particular, the occurrence of deadlines was accentuated as an issue that causes stress, as expressed by one participant: *"things happening near deadline is always more stressful than when you have a lot of time to fix"*.

4.2.3.4 Closed-ended Questions

As described in Section 3.8.1.1, some of the interview questions were closed-ended or concerned with validity issues. Those were excluded from the thematic analysis and will instead be presented here.

First, the frequency of affects are not necessarily constant, but changes over time. Certain periods, reportedly near deadlines, have a higher frequency of negative affects, such as stress, than others. In general, however, the participants typically estimate their affects, both negative and positive, to occur weekly or more frequently. As one interviewee answered: *"sometimes it happens once in three months and sometimes it may happen twice a week."*

Second, the participants did not report any external factors that might influence their responses. Some pointed out that their responses might be influenced because they, personally, were more engaged by software quality issues than their peers. Additionally, they reported that the situation itself might have impacted their responses, as expected due to the obtrusive and artificial nature of the session.

Third, the participants consistently reported that the software scenarios differed from industry software in that the problems were more isolated and lucid, but also that they were representative of the type of code and issues they might encounter in their daily work. As one participant stated: *"some of them were a bit extreme. Usually, it's more—no, not extreme—more localized. So it's like, it was really easy to identify the problem. In the industry it's more 'I know something is wrong. It feels like things are spread out like this. I just can't put my finger on it.'"*

Fourth, most participants did not find the SAM difficult to use or understand, especially when they had practised it in conjunction with the anchor point.

4.3 How does ATD impact the affects of software practitioners? (RQ1)

This research question seeks to address the relevance and importance of ATD to the affective state of software practitioners by analysing causal relationships between affects and design smells. In order to provide answers, the contrasts between the levels of each scenario were investigated for the three dimensions of the PAD framework. In other words, for each of the scenarios, the difference between responses for the high (H) and the low (L) level code example were analysed.

Figures 4.15-4.17 displays the distributions for the contrast of each scenario for valence, arousal, and dominance, respectively. The contrasts in *valence* (see Figure 4.15) for scenarios ScA, ScB, and ScE are centred close to zero, suggesting that the uncertainty is high concerning the effect. ScC is farther from the centre, with somewhere between 80 % and 95 % of the distribution to the left of the origin. Similarly, ScD is even farther from zero, with more than 99 % of the distribution to the right of the origin.

Using the 95 % threshold as a yardstick, we can see that only ScD has a confirmed effect on the affects of software practitioners. Since the direction of the comparison was high level of ATD minus low level of ATD, the results show that the occurrence of cyclically-dependent modularization design smells receive higher scores on the SAM for valence. Remembering that the SAM is designed such that low values represents affects such as pleasure, contentedness and hopefulness, and high values represent unhappiness, annoyance and unsatisfaction, these results show that yes, for certain smells, ATD does impact the valence of software practitioners.

Compared to valence, the effect of *arousal* is more uncertain. The contrasts (see Figure 4.16) for all scenarios are centred close to the origin, each with less than 80 % of the distributions to either side. Consequently, the data do not support the hypothesis that ATD would impact the arousal of software practitioners.

Similar to arousal, no conclusive statements can be made about the effect on *dominance*. Each of ScB, ScC, and ScE have distributions with less than 80 % outside of origio, while ScA and ScD have somewhere between 80 % and 95 % of their distribution to the side of zero. As a result, we cannot make any conclusive statements at the 95 % level of confidence about ATD causing a variation in the dominance of software practitioners.

In summary, there is compelling evidence that the scenario embodying the cyclically-dependent modularisation smell causes variations in affects among software practitioners. The participants who rated the version in which a small cyclic dependency occurred self-reported lower scores for valence, compared to those who rated the refactored version.

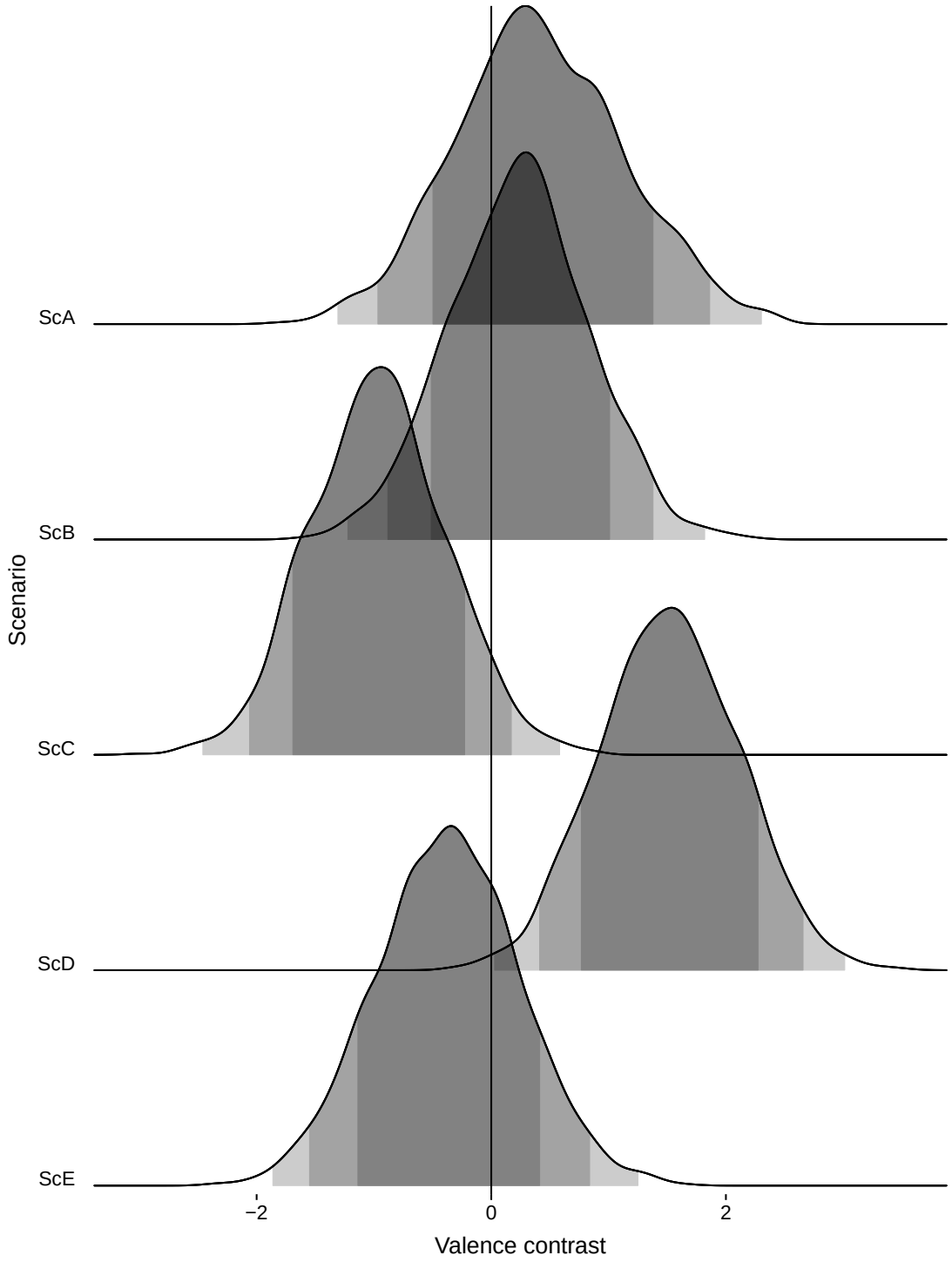


Figure 4.15: The contrasts for valence between high and low level versions of each scenario. Each distribution is marked with the 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs.

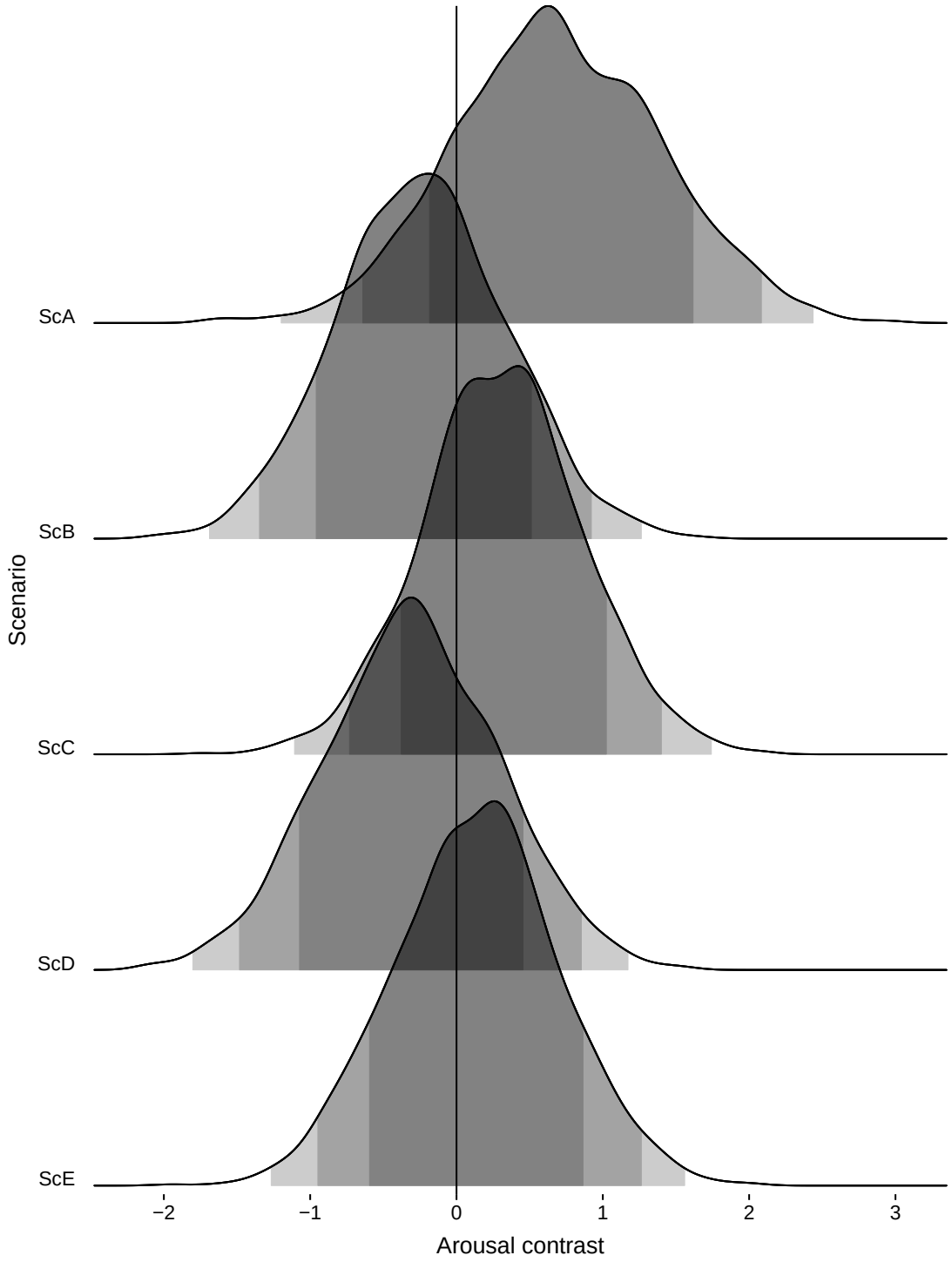


Figure 4.16: The contrasts for arousal between high and low level versions of each scenario. Each distribution is marked with the 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs.

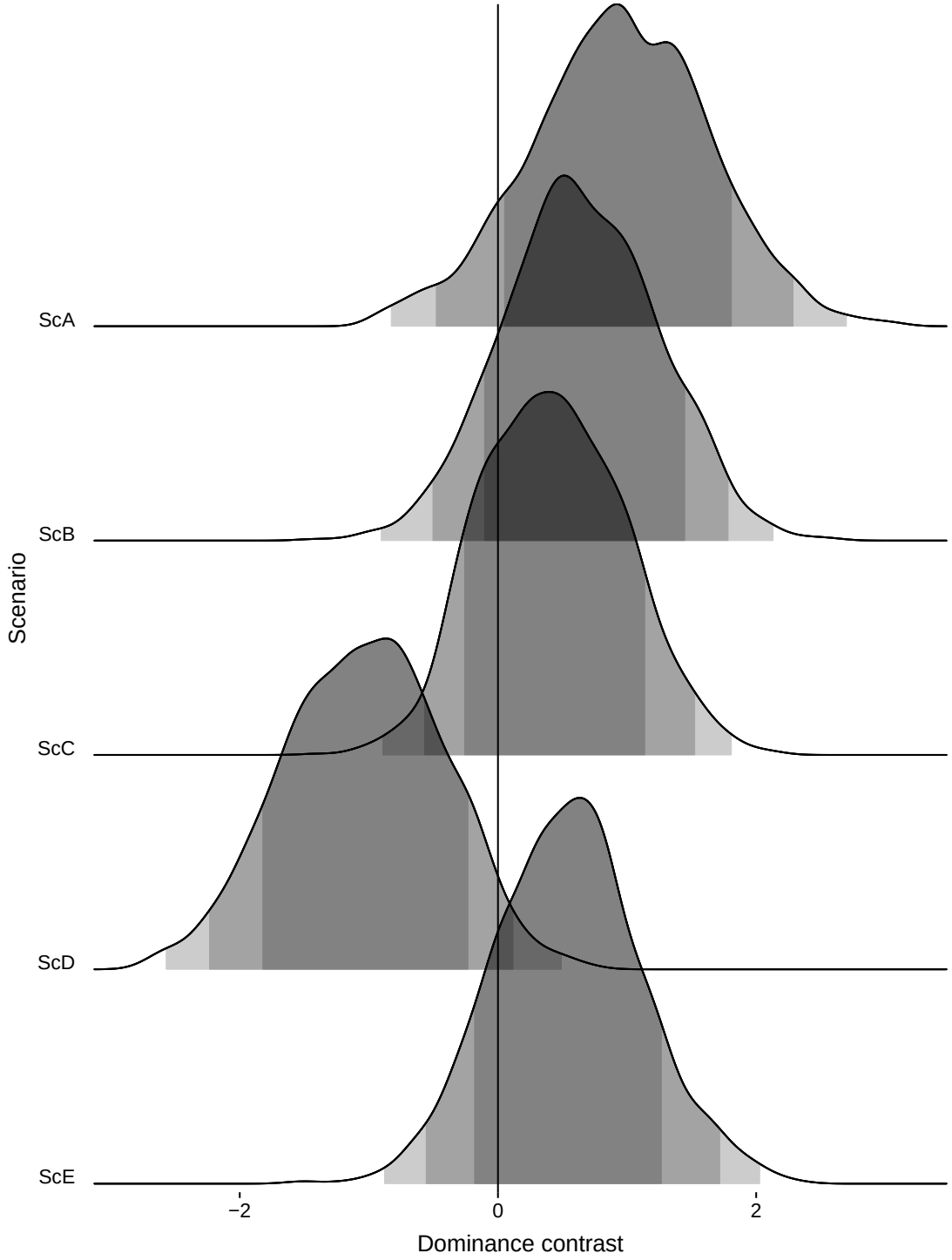


Figure 4.17: The contrasts for dominance between high and low level versions of each scenario. Each distribution is marked with the 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs.

4.4 How do professional characteristics correlate with changes in affects due to software smells? (RQ2)

In this section, we explore if professional background relates to changes in affects. Characteristics investigated are level of education, work experience, and work context. Each of these aspects are reported below, in their corresponding section.

4.4.1 Does formal education correlate with changes in affects due to design smells? (RQ2.1)

In order to answer RQ2.1, the extent of the participants' educations were investigated, as well as their majors. The data presented in Figure 4.18 shows that there is a large amount of uncertainty concerning the effect of level of education. Consequently, no conclusive statements can be made concerning whether or not the extent of one's education is related to one's affects toward the scenarios.

In Figures 4.19-4.21, the data concerning the effect of the participants' majors are shown. For each of the three dimensions, the distributions of every major are more or less centered at the origin. All of them deviate from zero with less than 80 % uncertainty and potential effects are therefore far from the 95 % threshold. As a result, we are unable to conclude that one's major is related to one's affective reaction to the scenarios.

To summarise, no evidence was found that would suggest that formal education is related to how software practitioners' affects vary due to design smells.

4.4.2 Does work experience correlate with changes in affects due to design smells? (RQ2.2)

RQ2.2 aims to reveal whether or not work experience plays a role in how the software practitioners react to the scenarios. In Figure 4.22, we can see that a large portion of the distributions are on different sides of the origin. This means that there is much uncertainty about the existence of an effect, and we can see that no conclusive statements can be made at the 95 % HDI.

In addition, we can see that the widths of the distributions are constrained to a comparatively small region. This means that even if there were an effect, its impact would have been close to negligible.

In answer to the research question, no evidence was found in support of work experience being related to changes in affects.

4.4.3 Does work context correlate with changes in affects due to design smells? (RQ2.3)

When examining the relationship between the work context and responses, the participants' work roles were investigated, as well as what programming language they were most experienced in.

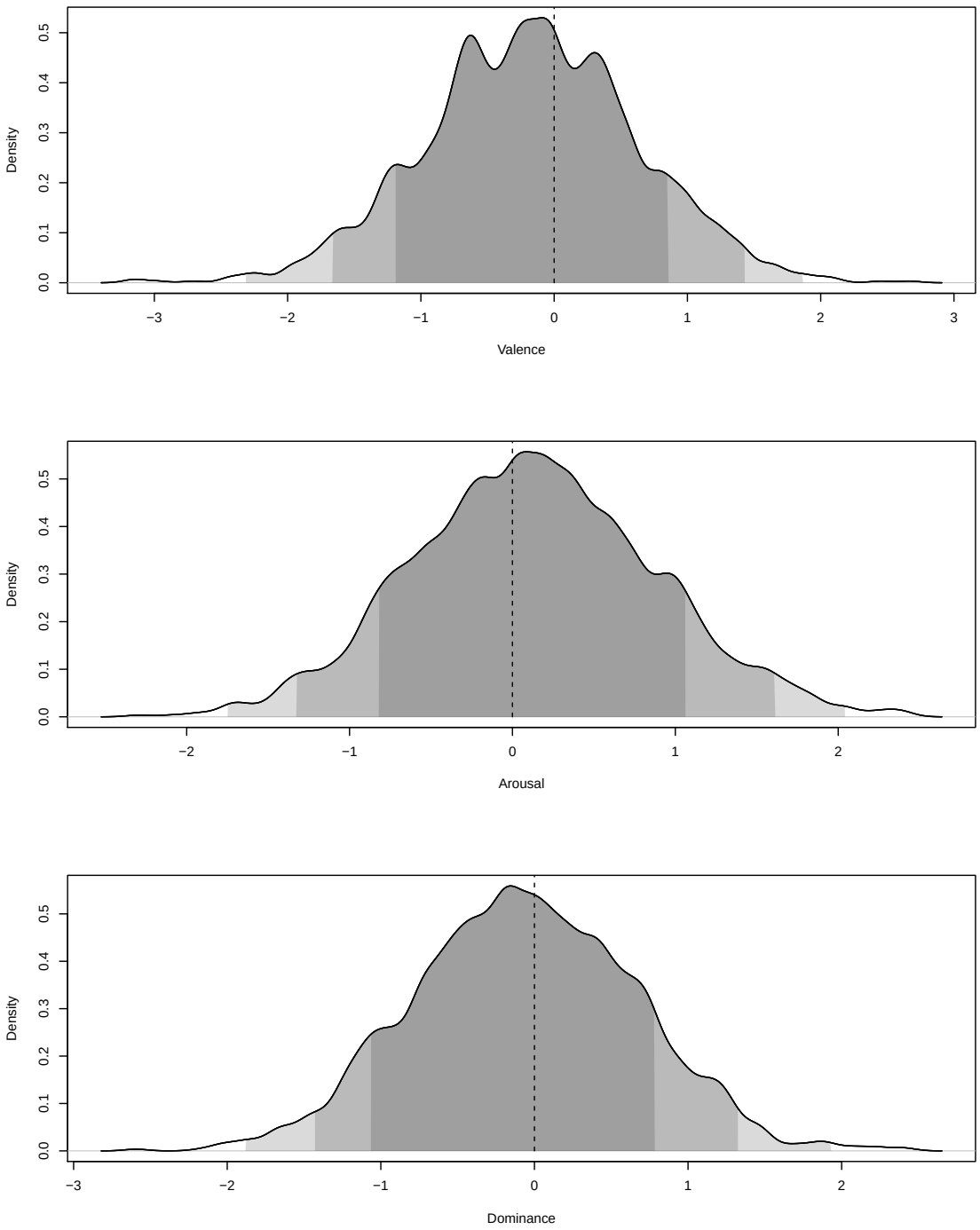


Figure 4.18: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of formal education on affective state responses.

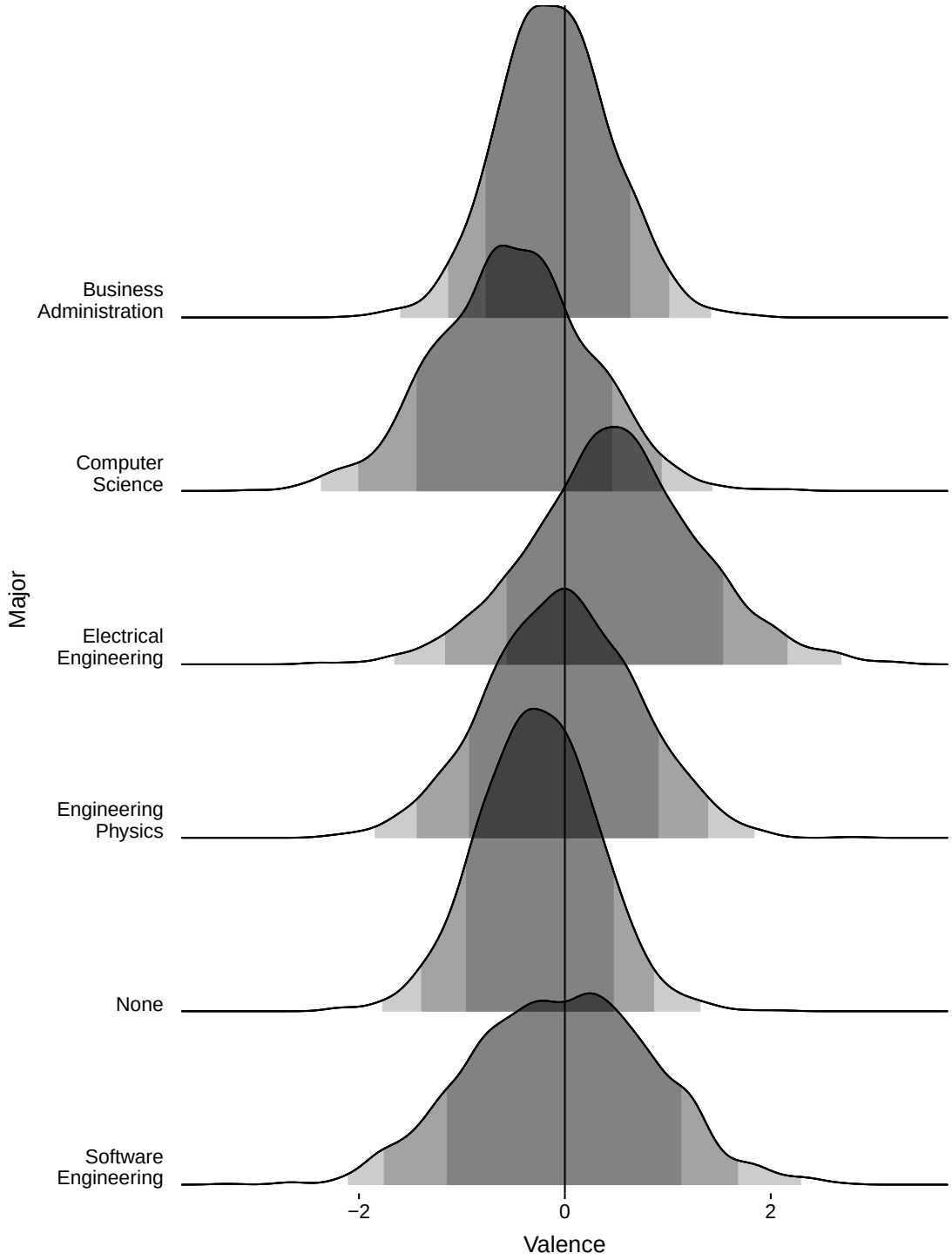


Figure 4.19: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of major on valence.

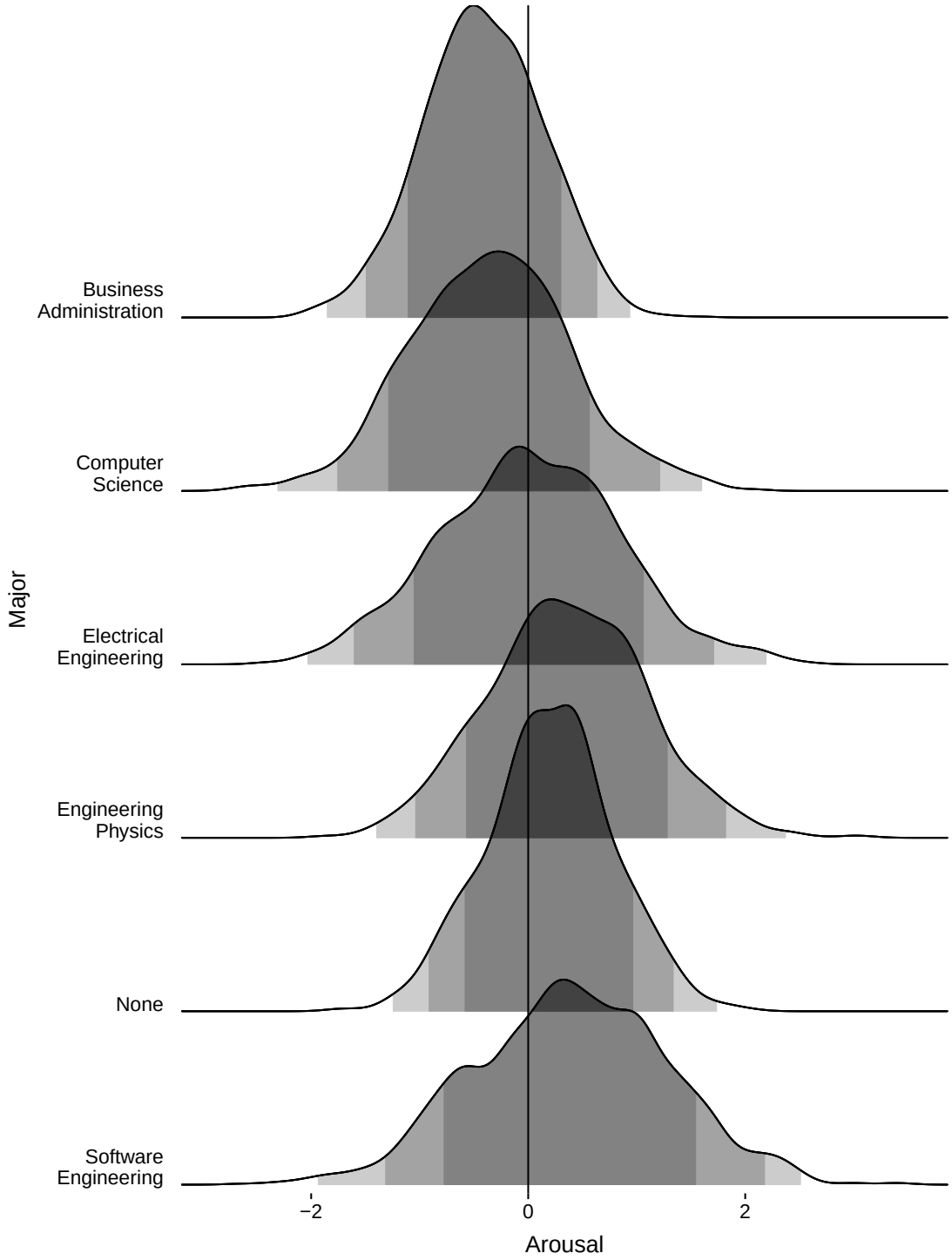


Figure 4.20: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of major on arousal.

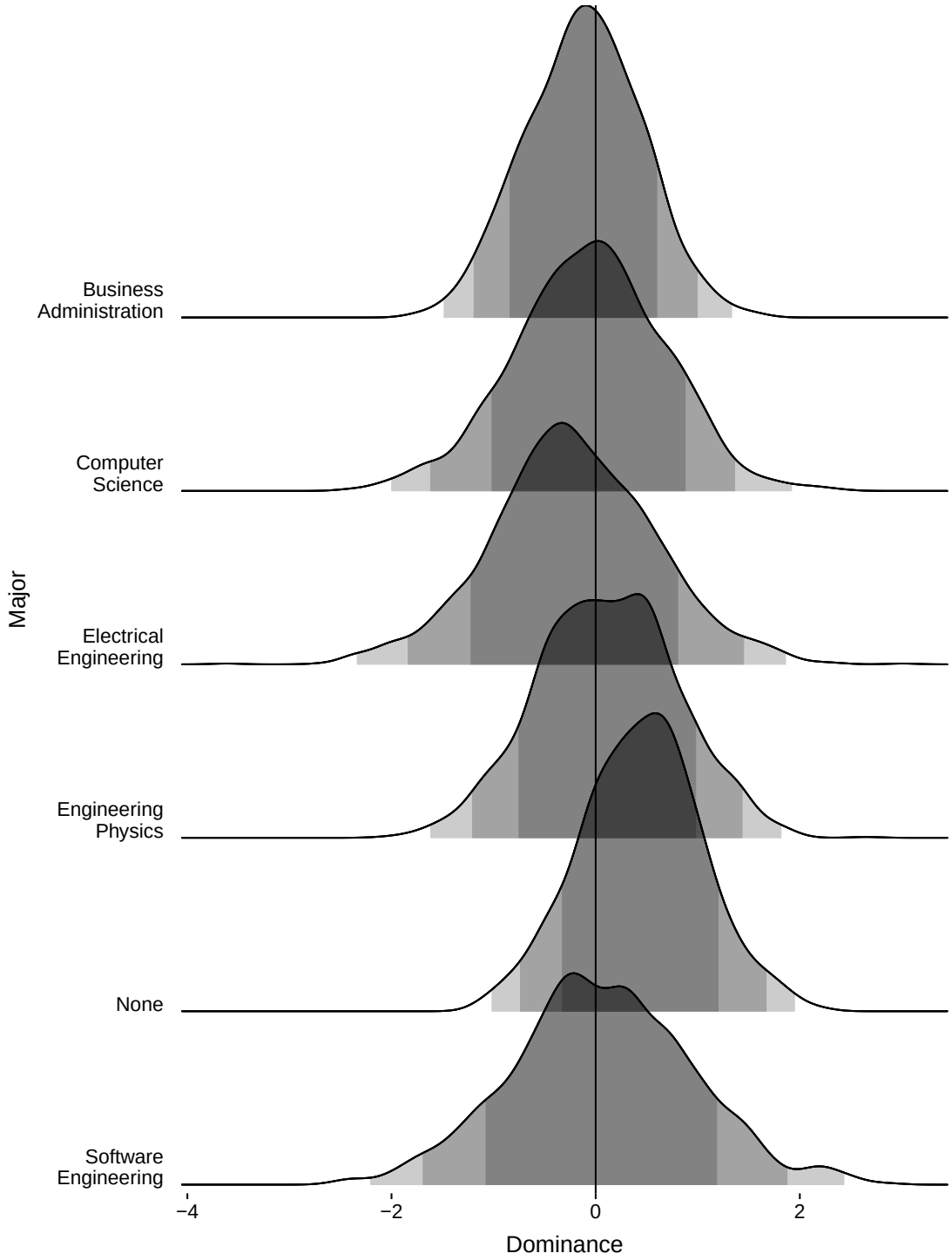


Figure 4.21: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of major on dominance.

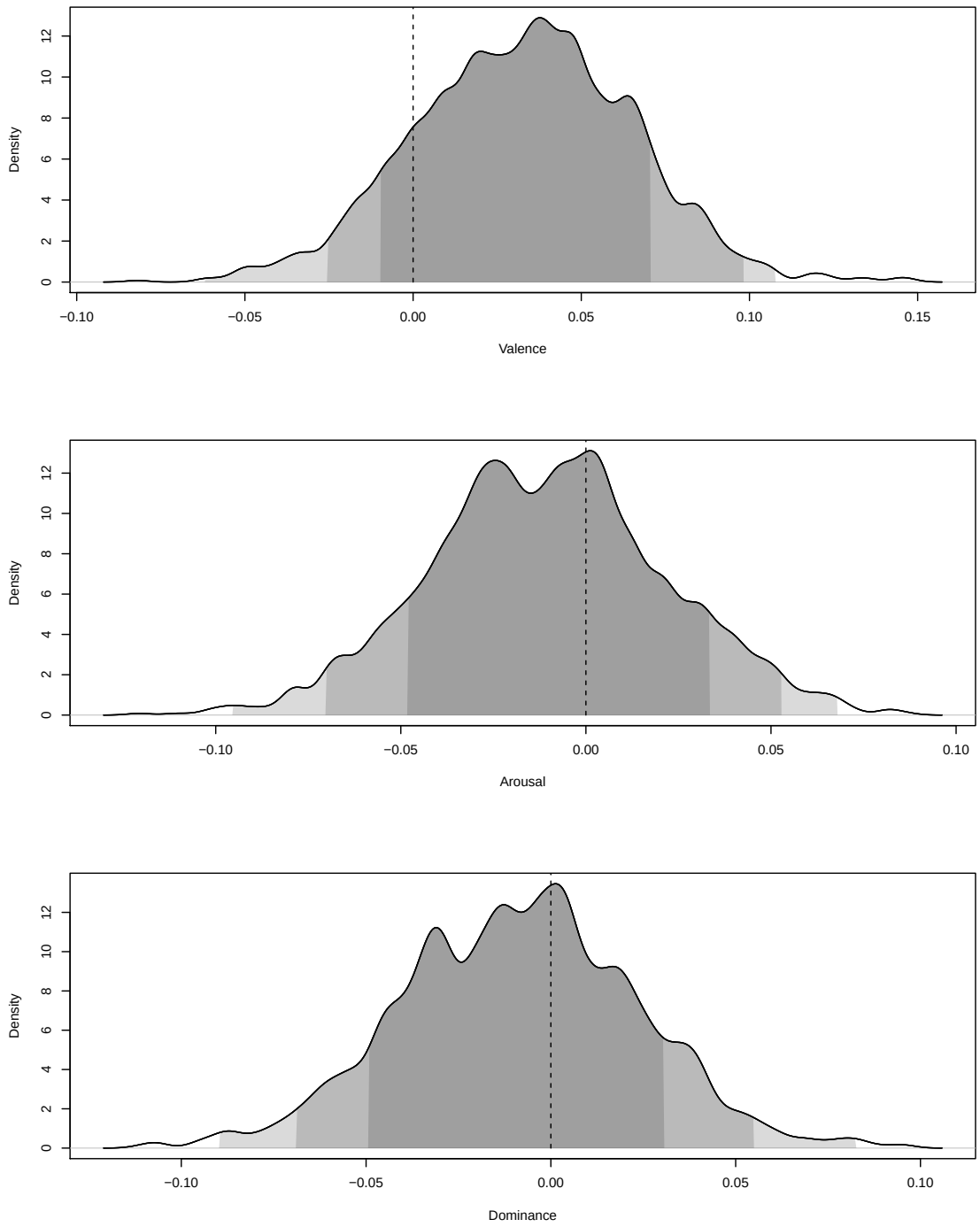


Figure 4.22: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of work experience on affective state responses.

Starting with work role, we see in Figures 4.23-4.25 that none of the roles convincingly indicate a change in affects at the 95 % level of certainty. All of the roles deviate from the origin with less than 80 % certainty for all dimensions, except for developers who barely pass the 80 % threshold for valence. Consequently, we are unable to conclude that one's work role is related to the affects one's emotional reaction to the scenarios.

For programming languages, there are effects with more than 95 % certainty for arousal and dominance, as illustrated in Figures 4.26-4.28. While most languages do not influence how the participant experienced the scenarios, practitioners with most experience in SAP ABAP consistently report lower scores on the arousal dimension of the SAM. Remembering that the scale is designed such that low values represent feeling excited and awake, and high values correspond to feeling relaxed, dull and unaroused, these results show that the programming language is related to higher levels of excitement. Similarly, the figure shows that practitioners most experienced in Ada report lower scores on the dominance dimension of the SAM, meaning that they have feelings characterised as controlled, influenced, cared-for, awed, submissive, or guided.

In short, there are indications that work context, in terms of what programming language the software practitioner is most experienced in, do relate to variations in affects due to design smells. The sole participant who was most experienced in SAP ABAP self-reported higher levels of arousal than others, while the two who were most experienced in Ada self-reported lower levels of dominance. As noted in Section 5.2, these findings are based on too few participants to be considered valid.

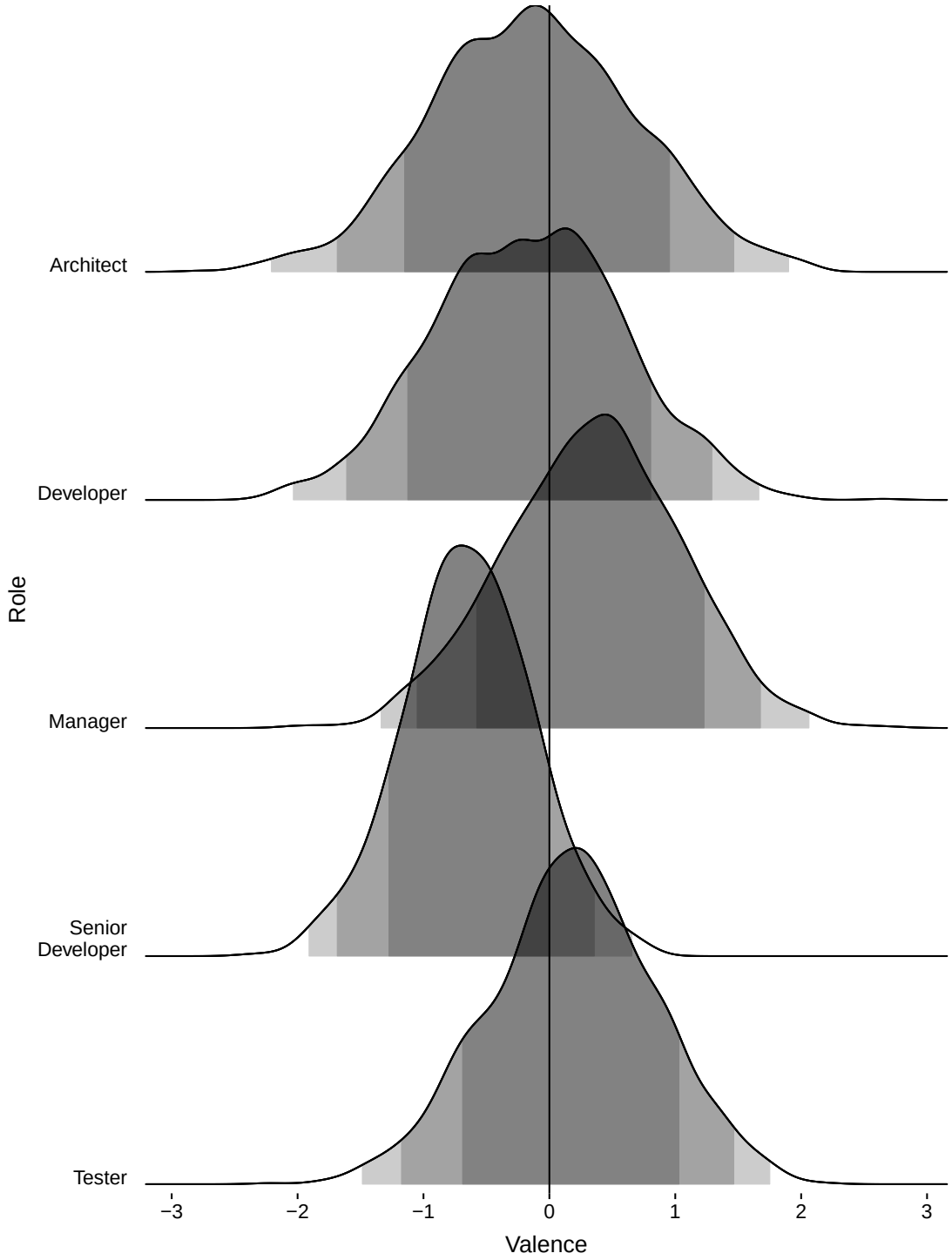


Figure 4.23: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of work role on valence.

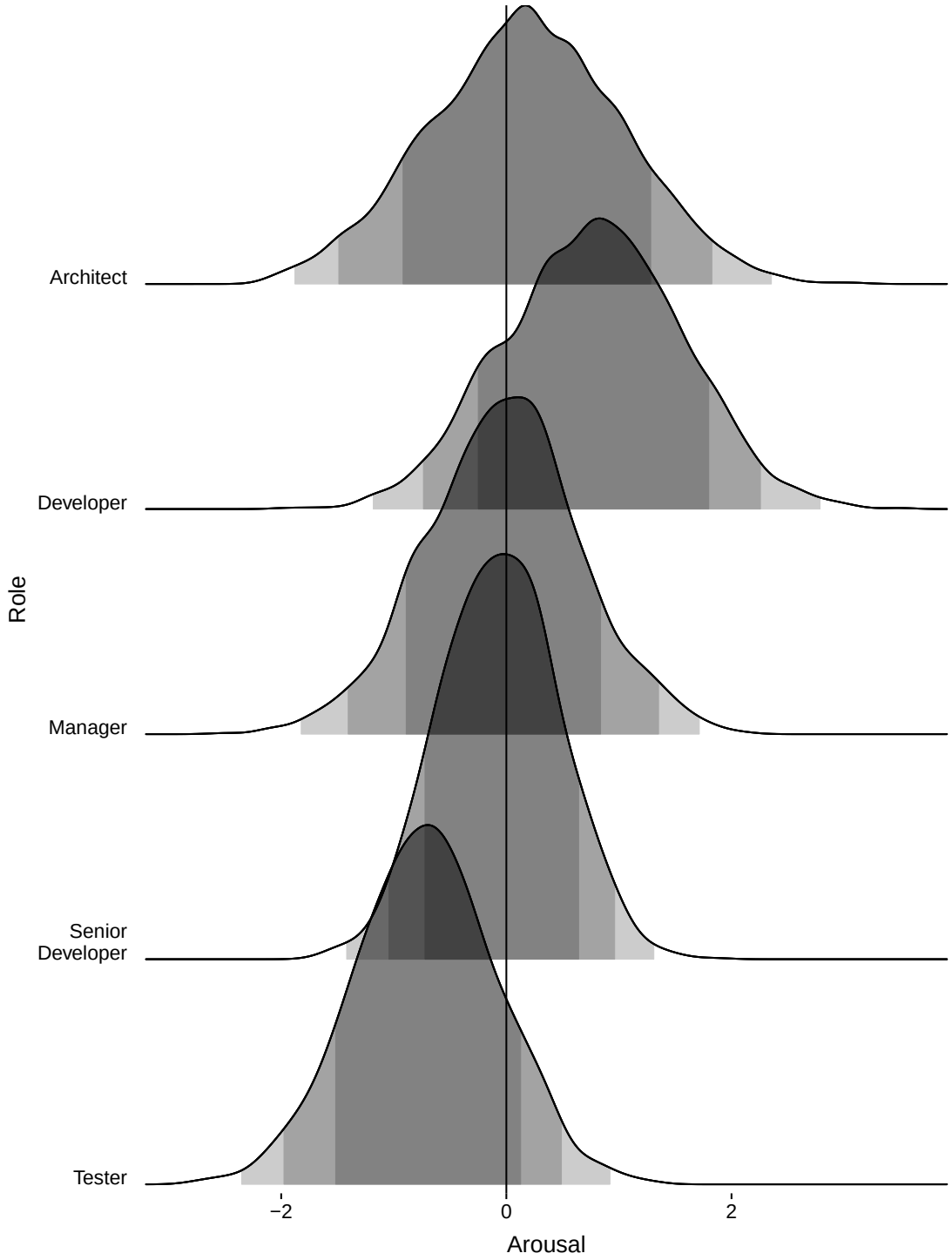


Figure 4.24: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of work role on arousal.

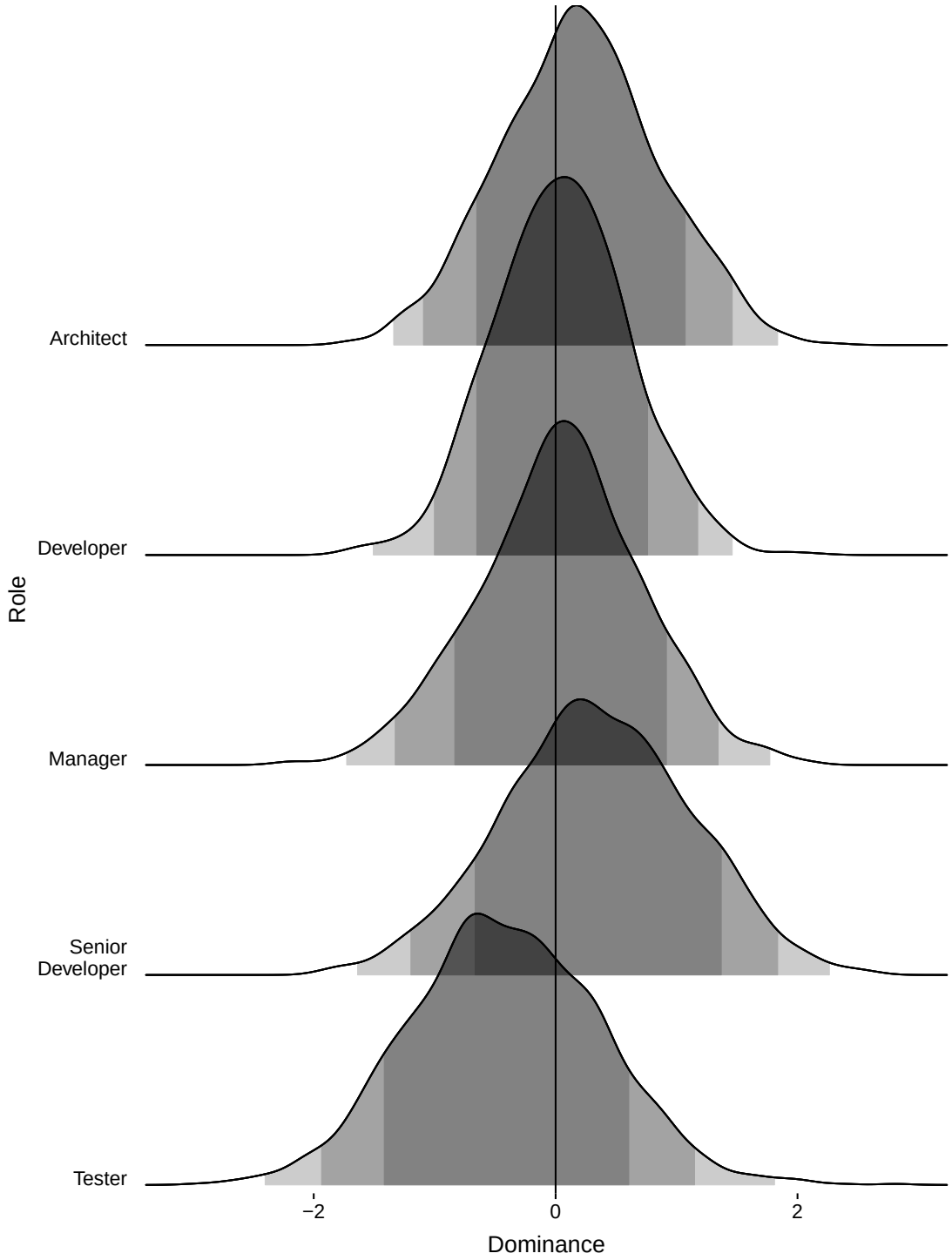


Figure 4.25: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of work role on dominance.

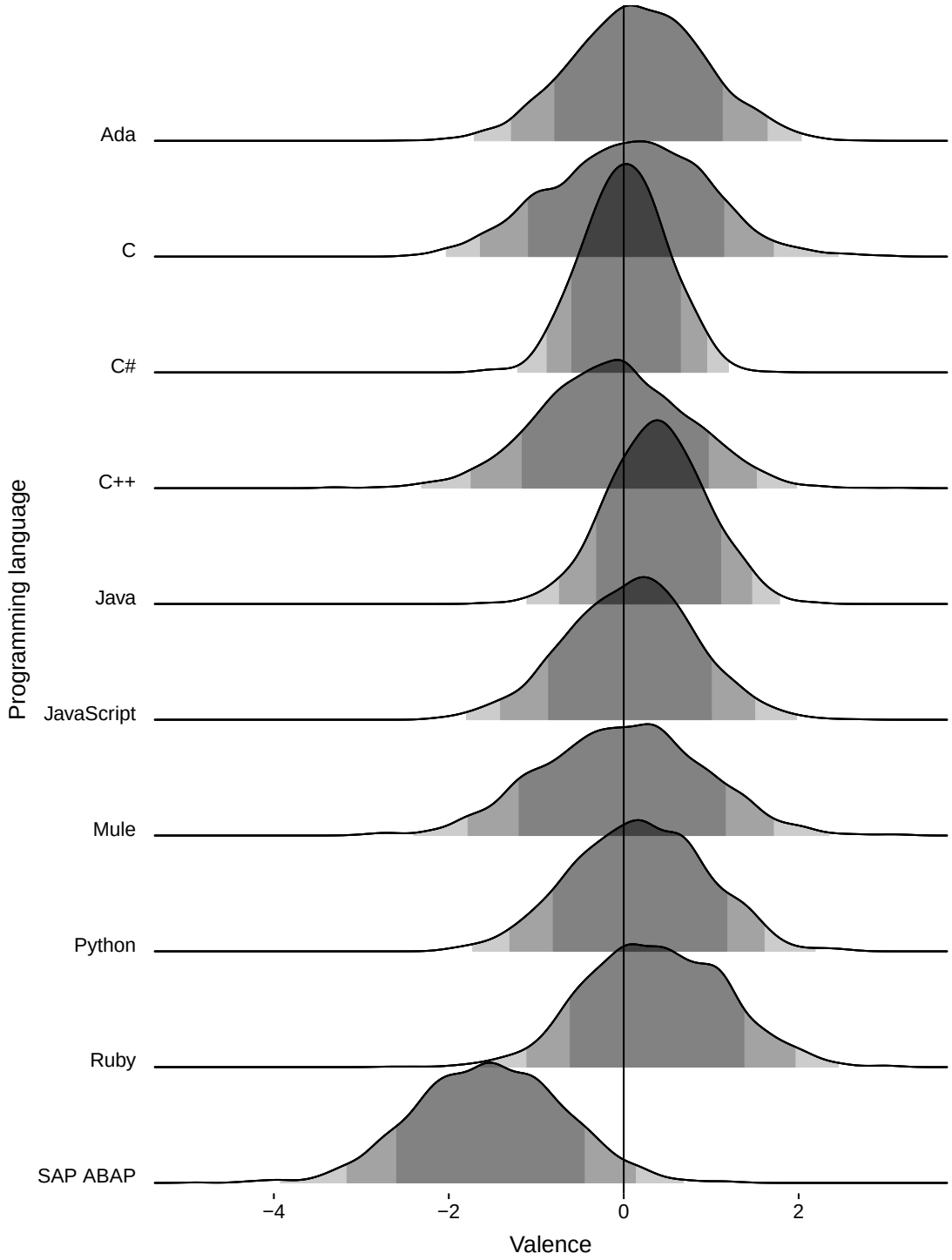


Figure 4.26: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of programming language most experienced in on valence.

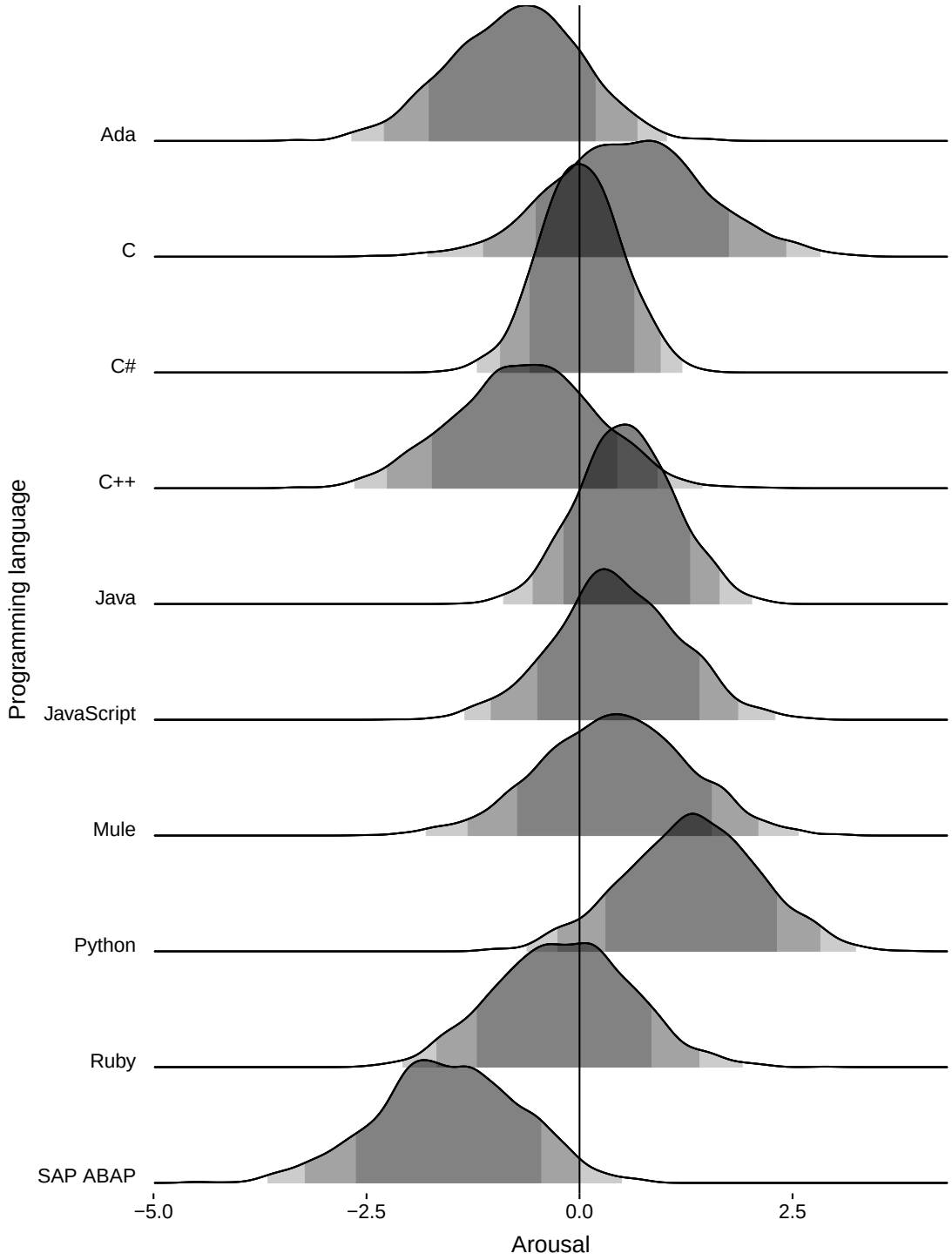


Figure 4.27: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of programming language most experienced in on arousal.

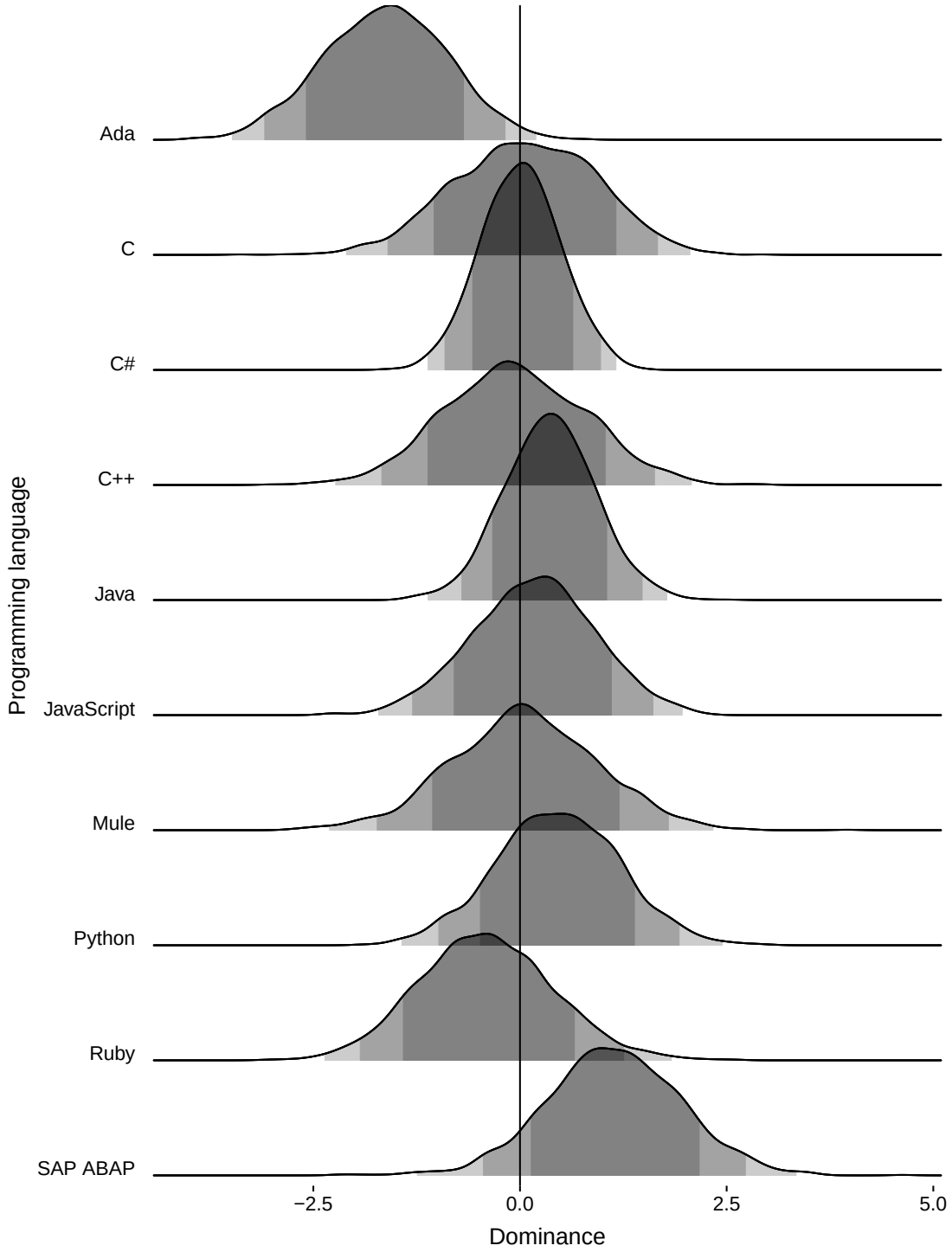


Figure 4.28: The 80 % (dark grey), 95 % (grey) and 99 % (light grey) HDIs for the effect of programming language most experienced in on dominance.

4.5 Do software practitioners attribute changes in affects to software engineering tasks? (RQ3)

This section presents the findings of software practitioners' affects in their daily work. Hence, these results go beyond the design of the laboratory experiment to report on peripheral and connected issues.

4.5.1 Do software practitioners readily recall instances where they have experienced changes in their affective state due to software engineering tasks? (RQ3.1)

This research question focuses on understanding how well-established the connection between software engineering tasks and affects are in the minds of software practitioners. From the thematic analysis, we found that the participants did readily recall various instances where software engineering tasks had impacted their affects, both positively and negatively. Indeed, both of those aspects were identified as main themes (see Section 4.2.3).

Starting with the positive affects, it was evident that the participants found pleasure in challenging tasks. The primary sources of these affects were credited to a sense of exploration or discovery. In particular, software practitioners seem to enjoy the sensation of learning something new and being able to make substantial improvements. As reported by one of the participants: *"That's what I like with this job, it's very challenging. You can actually push yourself into challenges and make stuff better all the time and rethink how it's done previously."*

Additionally, software practitioners find enjoyment in many specific software engineering activities. These encompass a variety of different dimensions of the software development process and as a group the participants reported finding positive experiences along the entire continua. They mentioned receiving positive affects from working with the software 1) on both high and low levels of abstractions; 2) from technical as well as social or managerial aspects; and 3) with a focus on enabling technologies or feature development.

Further, the outcome of their work is connected to positive affects. When able to see their tasks through to the end, software practitioners consider themselves productive and capable of solving problems. These concepts, in turn, were associated with accomplishment, encouragement, pleasure, and satisfaction. For example, one of the interviewees expressed that: *"it's nice to be able to check stuff off in the end of the day, feeling that you have accomplished something."*

Those affects are further emphasised when software practitioners perceive their work to be impactful, in particular if the quality of the project code base was improved.

The participants were divided on the topic of routine tasks, which was associated with both positive and negative affects. They were more frequently described in the negative than in the positive. Typically, routine tasks were considered boring, dull, and a waste of time. On the other hand, they were recognised as tasks that could be completed and thus would yield feelings of accomplishment, as mentioned above.

When analysing the negative affects of software practitioners a clear contrast to the positive ones was discovered. Positive affects were typically described with a component of agency, which was absent for the negative ones (see Section 4.2.3). This could be interpreted as though the participants to some extent correlate negative affects with a lack of dominance. In particular, the participants reported that they became hostile in situations where peers criticised their creative output. *"Yes, when somebody else is reviewing your code and have opinions on your code it can have spontaneous feelings, like, they are nitpicking or that they are wrong, because they don't like your beautiful code or they are not understanding."*

Among the sub-themes of negative affects, communication issues were frequently cited. The participants reported encountering several types of barriers that made them experience certain communication as annoying, stressful or frustrating. These affects were more potent when put in relation to time constraints, such as deadlines. Communication between different stakeholders was emphasised as a source of these problems. Reportedly, the difference in technical background leads to confusion and disagreements. As expressed by one participant: *"Communication is always a challenge. It's communicating between different parts of the teams and the company. And you have the business people that actually use the system, speaking in one way and the developers speaking in one way and the requirements team speaking one way and testing is somewhere between the requirement and the developers in the understanding of code. It's easier to speak with people that have experience in coding, because they understand it in a deeper way, what we are doing."*

The internal quality of the software was a source of negative affects that the participants described at length, and in more detail than other subthemes. It is clear that software practitioners receive a wide range of negative affects from poor code quality and that the issue pertains most, if not the entirety, of the industry.

At one end of the scale, the participants reported affects that were related to their role as professionals, rather than as private individuals. These affects were related to technical or professional aspects and expressed how the software quality might inhibit them in their work. These affects were chiefly concerned with annoyance, frustration, and aversion, and were the results of a technical context that decreased their productivity. As noted by one interviewee, there are issues that make the practitioners uncertain about, or distrust, the rest of the system. *"The person who designed the solution have thought about this in a fundamentally wrong way and that could be hard to actually fix these problems, since the whole design is based on that. It depends, it could be a small part of the application. Can refactor that, could be fine. But it's, it's like a warning signal. It's frustrating."*

In contrast, there were situations that induced more personal affects of sometimes large magnitude. The participants reported experiencing anger and hate from seeing code that they perceived as ugly or messy, and the use of profanity was not uncommon. These issues were found in many aspects of the system, but architectural issues, and in particular ripple effects, were more prevalent than others. As one of the participants highlighted, this type of technical context is demoralising and may cause software practitioners to refuse fixing defects. *"That makes you just angry inside, and when you work in it. Then it's like—even if it's me or my colleagues working in it—you can definitely feel when it's 'oh, I can refactor this' or 'uh, this*

is such a mess. I hate going into this code. I can't fix a bug here cause there is just going to pop up things in other places.'"

The final sub-theme to negative affects was that of time constraints, which was identified as a significant source of negative stress. While the participants acknowledged the need for time constraints, they reported that deadlines not only lead to stress and frustration, but also to significantly lower product and process quality. *"If you have, like... time pressure. That can also be technical debt, when you actually implement stuff. If you have time pressure to do something and then you also know that you're in—I mean, this is gonna be hard to test and to deliver it in time is gonna be tough. Then it's super stressful. But if you don't have that pressure again, then it's easier again."*

To summarise, the findings of the thematic analysis show that software practitioners receive both positive and negative affects from software engineering tasks and that ATD, in particular, reportedly is a source of strong negative affects.

4.5.2 Do software practitioners frequently experience affective state variations as a consequence of software engineering tasks? (RQ3.2)

The last research question sought to investigate if variations in affects from software engineering tasks are frequent and could result in long-term issues. The interviews show that the most common affects are positive. Among the participants who contrasted positive and negative affects, the positive ones were estimated to constitute between 75 % and 90 % of their affects. As one of the participants pointedly put it: *"I like what I do, otherwise I wouldn't be here."*

In Section 4.2.3.4, it was noted that the software practitioners' affects were clustered in periods of higher or lower frequency. Higher frequency periods co-align with deadlines, which can lead to burn out, as reported by one participant. *"Yeah, this was people that was sort of in the, more like architect roles, usually. Then they put on too much work on their shoulders and they were the guys that always wanted to do everything by themselves. And sort of tended to burn out after a while, because they just had too much to do and they, yeah. You could see that they were stressed about it."*

In conclusion, software practitioners' affects from software engineering tasks systematically suffer in the face of deadlines, but are otherwise predominantly positive.

5
Discussion

In this thesis, we conducted a psychoempirical software engineering study by employing quantitative and qualitative methods for investigating the relationship between technical debt and human aspects. This section will, in order, discuss the findings from the perspective of the three main research questions that were defined in Section 3.2. Additionally, the section will address threats to validity and consider implications for industry and academia.

5.1 How does ATD impact the affects of software practitioners? (RQ1)

Our results show that certain types of ATD do impact the affects of software practitioners. More precisely, our findings suggest that the design smell cyclically-dependent modularization lowers valence. While evaluated at 95 % confidence, this effect is also evident at the 99 % level. However, there are a couple of points concerning these results that would benefit from further discussion.

First, the 95 % per cent threshold, or any other arbitrary threshold, is a dangerous construct. As reported in Section 2.2.2, journal editors and numerous scientists are concerned about the use of preconceived thresholds to determine significance of results [47]. While it certainly is a reasonable stance to argue that research should hold up to some sort of standard, it is important to recognise that the problems investigated are diverse. It seems unreasonable to presume that a single common yardstick would see to the needs of them all.

Further, what should be considered significant likely also depends on one's viewpoint. In the scientific fields, findings have traditionally been ignored if the uncertainty of the result is more than 95 %, or 1/20. From an industry perspective, there may be reason to allow for a higher degree of uncertainty in certain situations. If the cost and effort are comparatively small and the potential payoff large, one might be willing to accept an uncertainty of 1/5 or even lower. Similarly, there are situations where a slight better-than-average chance is sufficient for action, such as initiating discussions about the problem within the own company. The reverse is also true, there could certainly exist situations where being wrong is too catastrophic to accept a proposition with an uncertainty of 5 %.

No matter what stance one takes, it is important to acknowledge that scientific results typically are not clear-cut. The circumstances and contexts in the large world make the problems multifaceted and commonly require and deserve more nuance than a binary answer.

For the sake of argument, consider the findings from the more permissive, but no less arbitrary, threshold of 80 %. This would yield the result that cyclically-dependent modularisation not only has a negative impact on the happiness of software practitioners, but also on dominance. Perhaps more interestingly, we also observe reverse outcomes for other types of ATD.

The participants had higher valence in response to the code example embodying the Broken Modularization smell than its refactored version. Likewise, the design smell Missing Encapsulation increased their dominance. These results and their implications are most interesting. While we have to confine to speculation, this could indicate that certain types of ATD pose a trade-off between internal software quality and the affects of software practitioners.

On the other hand, the source used for constructing those scenarios states that potential causes for missing encapsulations could be e.g. mixed up concerns or naive design decisions [23]. We also note that the refactoring introduces the Bridge software pattern, which is a high-level abstraction that is perhaps not commonly encountered by the average software practitioner. Under this premise, the variation in dominance could be explained due to a lack of acquaintance with the solution.

For broken modularisations, the author lists that potential causes for the smell is procedural thinking in an object-oriented context or a lack of knowledge of existing design [23]. While the majority of the respondents reported having most experience in object-oriented languages, namely C# and Java, this does not necessarily mean that procedural thinking is not eliminated from their working context. However, this seems less likely than neighbouring explanations, such that software practitioners prefer smaller, incorrectly decomposed entities to larger, correctly decomposed ones.

Second, because of the nature of this study, the scenarios were designed to be quickly understood. This makes them atypical instances of the ATD situations encountered in industry. Typically, ATD grows in an invisible and uncontrolled fashion and may procreate until the debt pertains large portions of the overall system. As a result, ATD scenarios encountered in the industry typically have higher principal and interest than our scenarios, as well as the source of the debt being less visible. This suggests that the data obtained in this study is an underestimation of the situation in the industry, where the effects are more tangible than shown in this study.

In spite of this, scenario ScD provided convincing evidence in favour of cyclic dependencies negatively impacting the valence of software practitioners. That scenario instantiated a small cyclic dependency of two nodes, i.e. a tiny tangle [60]. While frequent, such tangles are typically still easy to understand and work with [60, 23]. Presuming that the impact on valence is proportional to the size of the tangle, we can infer that situation in industry is considerably worse for the happiness of software practitioners.

In addition to size, the topology of the tangle might relate to how cyclic dependencies are experienced. Most tangles have irregular topologies [60] and it is believed that software practitioners perceive such tangles as more complex than those of highly regular topologies [60]. If we accept that valence is lower when the perceived complexity is higher, it is clear that the effects of our findings are larger in industry.

5.2 How do professional characteristics correlate with changes in affects due to software smells? (RQ2)

The collected data were unable to provide convincing evidence for a difference in affects between software practitioners of various professional backgrounds. Neither the level of formal education, major, work experience nor working role have clear indications on contributing to the reaction of the software scenarios. While the analysis suggests that what programming language the participants' had most experienced in correlates with their responses, the languages at issue were employed by a small portion of the sample. Consequently, the observed variations might be confounded by unmeasured factors of the individual subjects.

These findings are interesting, as they run counter to the beliefs of industry professionals, but corroborate previous studies. Ghanbari et al. reported that one third of their interviewees proposed that the scariness and frustration of working with heavily indebted legacy code are dependent on one's work experience [8]. Their findings did not support the claim [8], similar to how our findings were unable to detect work experience as an effect on valence, arousal, and dominance. These findings demonstrate the importance of empirically investigating the truthfulness of commonly held beliefs.

5.3 Do software practitioners attribute changes in affects to software engineering tasks? (RQ3)

The results of this study show that software practitioners do attribute changes in affects to software engineering tasks. They regard positive experiences as a characteristic of such tasks more frequently than negative ones and report that the frequency of affects is not constant over time.

Our findings suggest that managers should consider focusing on reducing sources of negative affects over providing sources of positive ones, as many software engineering tasks, at least to some degree, are intrinsically motivating and engaging. The participants readily recalled pleasure from challenging tasks and reported that completing tasks and making an impact made them feel productive and valuable. At the same time, the work context counteracted these affects. Poor software quality was thought of as an inhibitor that caused negative affects, sometimes as strong as hate. As reported by the participants, as well as in previous studies, such affects have concrete negative impacts on organisational performance [5].

It was noted that negative affects were mostly induced by maintainability issues, as opposed to other aspects of software quality. A common description of these instances was that a particular piece of a project code base was messy. This was understood as violations of architectural or design principles, such as modularity and acyclic dependencies. In other words, these findings suggest that the occurrence of ATD might be a crucial component of the negative affects experienced from software engineering tasks. Consequently, the expected cost of ATD is, with a high degree of

certainty, higher than indicated by previous research, due to its impact on human aspects. We therefore issue a call for further research on the topic, with a focus on estimating the financial cost of those consequences. We also recommend the software industry to include these aspects, although not fully valorised, in their risk analyses of TD.

The use of, and need for, deadlines would benefit from reevaluation, in order to assess if the time-to-market benefits compensate for the drawbacks. The participants reported that time constraints not only decreases product and process quality, but also bolster negative affects in both frequency and magnitude. At times, deadlines cause strong negative stress and burnout. Given the high demand for software practitioners and the positive affects provided by software engineering tasks, it is reasonable to presume that companies with an abundance of deadlines will bleed talent.

5.4 Threats to validity

Conducting empirical studies with human subjects is often a complex issue [55], as it is often the case that the context is noisy and the effects investigated small [43]. As such, the potential threats to validity are often numerous and it would be infeasible to fully discuss them all. In this section, we present what we consider the most significant validity threats of this study and measures taken to mitigate them. The threats are categorised according to the aspects suggested by [61] and [62].

5.4.1 Construct validity

This study set out with the aim of determining how ATD impacts the effects of software practitioners, in part through laboratory experiments. As mentioned in Section 3.7.1.4, the subjects were presented with five software design smells or their respective refactored version. However, those smells were instantiated in code examples that originated from the same source, namely [23], which is neither a scientific journal article nor a conference paper. In addition, the purpose of the source differs from ours, in that the smells and the refactored versions are intended to be contrasted with each other. In the same section, we gave an account for the rationale of this choice: there seems to be a deficit of concrete ATD representations in the research literature, at least examples that would be suitable for the context of this study.

These characteristics do introduce threats to validity, but because they were identified prior to the data collection, countermeasures could be introduced. Since we were unable to find a way to eliminate these threats, we chose to monitor them and investigate the issue post-hoc. This was done by introducing question IQ4.1, IQ4.2 and IQ5 (see Section 3.8.1.1) and analysing the answers.

As reported in Section 4.2.3.4, the participants confirmed that the scenarios were representative industry code, albeit atypically small and isolated examples of TD encountered in practice. This suggests that the treatment was suitable, but also that the resulting data is an underestimation of the situation in the industry.

5.4.2 Internal Validity

The laboratory experiment part of this study was repeated measures design. While this approach mitigates the threat of confounding factors, because the peculiarities of each subject are accounted for, it is more susceptible to learning effects.

Several countermeasures were taken in order to reduce learning effects. First, the participants were acquainted with the situation during the pre-task instructions (see Section 3.6) and each received the same instructions for how to use the measurement instrument and their task. Second, the participants obtained practical experience with the procedure before the measurements, as a result of the anchor point (Section 3.7.1.2). Third, intermissions were used between measurements (see Section 3.7.1.3 in an attempt to lower the probability of any affects induced by previous scenarios were carried over to later ones. Fourth, each participant was randomly allocated to one of two complimentary treatments patterns (see Section 3.7.1.1), which were designed to minimise bias. Fifth, and perhaps most importantly, the order of the scenarios were randomised for each participant.

5.4.3 External Validity

As described in Section 2.2.2, the field of psychology is experiencing a replication crisis. Unfortunately, we have been unable to find consensus on concrete best practices for ensuring replicability and have instead chosen adopt some of the propositions.

We have made our work as transparent as possible, under the constraints set by confidentiality and anonymity. This means that we, in addition to results of the statistical analysis, has published the associated data (see Appendix I). In the ideal case, the data used for the analysis would be identical to the collected data, but as detailed in Section 4.2 some information had to be reduced. Similarly, we have described the procedure at length and made the experiment material available (see Appendices D-H)

Another issue concerned with external validity is the sampling strategy. In this study, we employed convenience sampling (see Section 3.3) by using specific professional networks. The approach meant that the sample was limited in several ways. First, all participants were industry professionals, which is a subset of all software practitioners and might not be representative. Second, the participants were selected by their managers, who might have their own agenda in what employees we, as soon-to-graduate software engineering students, met. Third, the companies belonged to the subset of companies that is both sufficiently interested in this study to take part and covered by the combined professional networks.

However, our results show that the data is inconclusive concerning the effects of different professional characteristics, such as work experience and role. This could indicate that the study is less susceptible to convenience sampling than otherwise. Further, as described in Section 4.2, the 40 participants had a wide variety of professional backgrounds and were employed at 12 different companies.

Along the same line, the generalisability of the results of the study is threatened by demographic factors. Due to various constraints, including financial, all partaking companies had offices in Sweden. While the study was conducted in several parts of the country, Sweden as a whole is culturally distinguished in terms of secular-rational

and self-expression values [63]. That said, there was diversity in ethnicity among the participants, but such data was not collected in order to protect confidentiality. For the same reason, many aspects of the participants' demographic profiles were not investigated.

5.4.4 Conclusion Validity

The investigation undertaken in this thesis is novel in the sense that, as far as we can tell, no previous studies have investigated how TD impacts affects (see Section 2.3). Consequently, the findings of this study cannot be compared and contrasted with the findings of others. Instead, they must be evaluated in isolation and are, therefore, more susceptible to incorrect inferences and conclusions.

To combat these threats, three triangulation techniques [55] were adopted. First, the data were triangulated in the sense that the sessions were spread out over four weeks and the participants were employed at different companies. Second, researcher triangulation was achieved as both authors took part in all data gathering and interpretation. Third, methodological triangulation was used, as the data were collected through experiments, a questionnaire, and an interview.

5.5 Implications for Industry

In this study, we have discovered instances where ATD impacts the affects of software practitioners. With more than 99 % certainty, our data shows that even tiny cyclic dependencies make software practitioners more unhappy. This could have widespread and far-reaching ramifications for the software industry, some of which we will discuss in this section.

Software engineering is a human activity, which makes it part of a complex network of societal, economical, and political issues. This introduces a large number of biases that make the activity vulnerable to a wide range of issues. Some of those are related to ATD and the management thereof.

Other researchers have found that unhappy developers, in addition to 46 other consequences, exhibit lower cognitive performance, have lower motivation, and produce lower quality code [5]. It is reasonable to assume that those consequences coalesce in unhappy software practitioners incurring more or, perhaps more accurately, more detrimental instances of ATD than their happy colleagues. Combine this with the results of this study and we see the seeds sown for vicious cycles of ATD.

Put differently, there is a risk that the human aspects of ATD will increase the amount of the debt of its own accord. If this issue is left unmanaged, the effort to make future changes would compound and the cost of would thus increase non-linearly. As we can see, the impact of ATD on the affects of software practitioners can, independent of the technical aspects, cause severe and tangible financial risks for companies developing software.

Clearly, there are technological reasons for companies to include and valorise the human aspect in any ATD risk analyses. In addition to those reasons, there are also human resource management aspects. The typical situation in the industry

is that there is high demand for, but low supply of, software practitioners. Consequently, the software practitioners are valuable employees that are both difficult and expensive to replace.

Our findings indicate that software practitioners would not put it past themselves to change jobs because of the quality of the code base in general, and the amount of TD in particular. In addition to the potential of higher turnover rate and talent bleed, the fact that ATD makes software practitioners unhappy lowers their productivity [5].

Neither this study nor the body of knowledge is able to provide guidance in assessing those values as there are no known valorisations of the impact of ATD on affects. This means that decision-makers are left with a large amount of uncertainty in how to deal with these implications. Because of this, we make a call for additional research, to build upon the findings of this study on this topic to further elucidate the complex relationship between human aspects and ATD.

6
Conclusion

Previous research links the affects of software practitioners to many issues concerning work performance and product quality. In spite of this, few studies have been conducted on the topic within technical debt research. Although evidence suggests that developers' morale is impacted by technical debt, the impact on human aspects has yet to be acknowledged as a consequence of technical debt.

This study set out to empirically investigate causal relationships between architectural technical debt and the affects of software practitioners. In order to archive this objective, a mixed methods approach was employed, which combined laboratory experiments and semi-structured interviews. 40 participants from 12 companies attended 90-minute sessions during which they used the Self-Assessment Manikin to rate their affects in response to five software scenarios and answered questions about their professional background and their experience with affects in relation to software engineering.

The resulting data set is rich with over 200 data points. It provides evidence beyond 99 % certainty that even tiny circular dependencies lower the valence of software practitioners. In addition, the participants reported that software scenarios were less impactful than what they encounter in practice, suggesting that our findings provide a lower bound on the situation in the industry.

The contribution of this thesis is threefold. First, we have drawn from the literature on psychoempirical software engineering to formulate a successful procedure for post-hoc measurement of the consequences of various technical debt items on the affects of software practitioners, which can be recycled for future investigations in both academia and industry.

Second, we have collected and published a rich data set as open data for use in further studies. The data set contains 200 measurements of how the affective state of software practitioners of various professional backgrounds is impacted by five types of software design smells.

Third, we have accentuated the consequences of architectural technical debt on human aspects of industry professionals by providing convincing empirical quantitative and qualitative evidence on the topic. The findings show that the occurrence of even tiny tangles makes software practitioners more unhappy, with more than 99 % certainty. Further, we were unable to find any differences in the affects based on various professional characteristics. Moreover, the results suggest that architectural technical debt encountered in practice cause intense negative affects and are bolstered by time constraints.

The results provide decision makers with an additional dimension to consider in technical debt management. The investigation has also contributed to a deficit area

in software engineering research and provides strong arguments for extending the conceptual model of technical debt to include consequences on human aspects.

The combined quantitative and qualitative data suggest several promising avenues for future research. First, there are indications that our findings would become more articulated in the context of large-scale software systems, compared to the small and isolated software scenarios exercised in this study. Second, a quantification of the human resources risks associated with architectural technical debt would nuance the research on technical debt management. Third, the characteristics and consequences of project code bases that software practitioners perceive as messy should be investigated further, as they appear detrimental for the well-being of software practitioners.

Bibliography

[1] W. Cunningham, "The WyCash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[2] S. M. Kompaso and M. S. Sridevi, "Employee engagement: The key to improving performance," *International journal of business and management*, vol. 5, no. 12, p. 89, 2010.

[3] I. A. Khan, W.-P. Brinkman, and R. M. Hierons, "Do moods affect programmers' debug performance?" *Cognition, Technology & Work*, vol. 13, no. 4, pp. 245–258, 2011.

[4] D. Graziotin, X. Wang, and P. Abrahamsson, "Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering," *Journal of Software: Evolution and Process*, vol. 27, no. 7, pp. 467–487, 2015.

[5] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "Unhappy developers: Bad for themselves, bad for process, and bad for software product," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 362–364.

[6] D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, vol. 2, p. e289, 2014.

[7] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt-When and by whom will it be paid?" in *IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China*, 2017.

[8] H. Ghanbari, T. Besker, A. Martini, and J. Bosch, "Looking for Peace of Mind? Manage your (Technical) Debt - An Exploratory Field Study," in *11th International Symposium On Empirical Software Engineering and Measurement (ESEM), Toronto, Canada*, 2017.

[9] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[10] W. Cunningham, "Ward Explains Debt Metaphor," 2009. [Online]. Available: http://wiki.c2.com/?WardExplainsDebtMetaphor

[11] R. C. Martin, "A Mess is not a Technical Debt. - Clean Coder," 2009. [Online]. Available: https://sites.google.com/site/unclebobconsultingllc/a-mess-is-not-a-technical-debt

[12] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6693

[13] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Information and Software Technology*, vol. 64, pp. 52–73, 2015.

[14] M. Fowler, "Technical Debt Quadrant," 2009. [Online]. Available: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[15] S. McConnell, "Managing technical debt," *Construx Software Builders, Inc*, pp. 1–14, 2008.

[16] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121213000022

[17] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[18] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Sp\'\inola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Information and Software Technology*, vol. 70, pp. 100–121, 2016.

[19] T. Besker, A. Martini, and J. Bosch, "Managing architectural technical debt: A unified model and systematic literature review," *Journal of Systems and Software*, vol. 135, pp. 1–16, jan 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121217302121

[20] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Sp\'\inola, "Towards an ontology of terms on technical debt," in *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 1–7.

[21] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158–173, 2018.

[22] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Toward a catalogue of architectural bad smells," in *International Conference on the Quality of Software Architectures*. Springer, 2009, pp. 146–162.

[23] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann, 2014.

[24] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, and E. Di Nitto, "Arcan: a tool for architectural smells detection," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 282–285.

[25] S. Cruz, F. Q. da Silva, and L. F. Capretz, "Forty years of research on personality in software engineering: A mapping study," *Computers in Human Behavior*, vol. 46, pp. 94–113, 2015.

[26] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE transactions on software engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[27] R. Feldt, L. Angelis, R. Torkar, and M. Samuelsson, "Links between the personalities, views and attitudes of software engineers," *Information and Software Technology*, vol. 52, no. 6, pp. 611–624, 2010.

[28] R. Colomo-Palacios, A. Hernández-López, Á. García-Crespo, and P. Soto-Acosta, "A study of emotions in requirements engineering," in *World Summit on Knowledge Society*. Springer, 2010, pp. 1–7.

[29] D. Graziotin, X. Wang, and P. Abrahamsson, "Software Developers, Moods, Emotions, and Performance," *IEEE Software*, vol. 31, pp. 24–27, 2014.

[30] P. Lenberg, R. Feldt, and L. G. Wallgren, "Behavioral software engineering: A definition and systematic literature review," *Journal of Systems and Software*, vol. 107, pp. 15–37, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000989

[31] D. Graziotin, X. Wang, and P. Abrahamsson, "Understanding the affect of developers: theoretical background and guidelines for psychoempirical software engineering," in *Proceedings of the 7th International Workshop on Social Software Engineering*. ACM, 2015, pp. 25–32.

[32] J. A. Russell and A. Mehrabian, "Evidence for a three-factor theory of emotions," *Journal of Research in Personality*, vol. 11, no. 3, pp. 273–294, sep 1977.

[33] B. N. Lang, Peter J and Bradley, Margaret M and Cuthbert, "International affective picture system (IAPS): Technical manual and affective ratings," *NIMH Center for the Study of Emotion and Attention*, vol. 1, pp. 39—-58, 1997.

[34] D. Graziotin, X. Wang, and P. Abrahamsson, "The affect of software developers: common misconceptions and measurements," in *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE Press, 2015, pp. 123–124.

[35] C. D. Fisher and N. M. Ashkanasy, "The emerging role of emotions in work life: An introduction," *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior*, vol. 21, no. 2, pp. 123–129, 2000.

[36] P. J. Lang, "Behavioral treatment and bio-behavioral assessment: Computer applications," *Technology in mental health care delivery systems*, pp. 119–137, 1980. [Online]. Available: http://www.citeulike.org/user/acagamic/article/3756983

[37] M. M. Bradley and P. J. Lang, "Measuring emotion: the self-assessment manikin and the semantic differential," *Journal of behavior therapy and experimental psychiatry*, vol. 25, no. 1, pp. 49–59, 1994.

[38] J. D. Morris, C. Woo, J. A. Geason, and J. Kim, "The power of affect: Predicting intention," *Journal of Advertising Research*, vol. 42, no. 3, pp. 7–17, 2002.

[39] J. D. Morris, "Observations: SAM: The self-assessment manikin: An efficient cross-cultural measurement of emotional response." *Journal of Advertising Research*, 1995.

[40] A. Betella and P. F. M. J. Verschure, "The Affective Slider: A Digital Self-Assessment Scale for the Measurement of Human Emotions," *PLOS ONE*, vol. 11, no. 2, p. e0148037, feb 2016. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0148037

[41] H. Pashler and E.-J. Wagenmakers, "Editors' introduction to the special section on replicability in psychological science: A crisis of confidence?" *Perspectives on Psychological Science*, vol. 7, no. 6, pp. 528–530, 2012.

[42] O. S. Collaboration and Others, "Estimating the reproducibility of psychological science," *Science*, vol. 349, no. 6251, p. aac4716, 2015.

[43] A. Gelman, "The failure of null hypothesis significance testing when studying incremental changes, and what to do about it," *Personality and Social Psychology Bulletin*, vol. 44, no. 1, pp. 16–23, 2018.

[44] Z. Dienes and N. McLatchie, "Four reasons to prefer Bayesian over orthodox statistical analyses," *Psychonomic Bulletin and Review*, 2016.

[45] R. McElreath, *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018.

[46] A. Ortega and G. Navarrete, "Bayesian Hypothesis Testing: An Alternative to Null Hypothesis Significance Testing (NHST) in Psychology and Social Sciences," in *Bayesian inference*. IntechOpen, 2017.

[47] V. Amrhein, S. Greenland, and B. McShane, "Scientists rise up against statistical significance," 2019.

[48] C. Fernández-Sánchez, J. Garbajosa, A. Yagüe, and J. Perez, "Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study," *Journal of Systems and Software*, vol. 124, pp. 22–38, 2017.

[49] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE software*, vol. 29, no. 6, pp. 22–27, 2012.

[50] J. Yli-Huumo, A. Maglyas, and K. Smolander, "The sources and approaches to management of technical debt: a case study of two product lines in a middle-size finnish software company," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2014, pp. 93–107.

[51] R. O. Spínola, A. Vetrò, N. Zazworka, C. Seaman, and F. Shull, "Investigating technical debt folklore: Shedding some light on technical debt opinion," in *2013 4th International Workshop on Managing Technical Debt (MTD)*, may 2013, pp. 1–7.

[52] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "On the unhappiness of software developers," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2017, pp. 324–333.

[53] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[54] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 275–284.

[55] J. Miller, "Triangulation as a basis for knowledge discovery in software engineering," *Empirical Software Engineering*, vol. 13, no. 2, pp. 223–228, 2008.

[56] P. Gomez, P. G. Zimmermann, S. Guttormsen Schär, and B. Danuser, "Valence lasts longer than arousal: Persistence of induced moods as assessed by

psychophysiological measures," *Journal of Psychophysiology*, vol. 23, no. 1, pp. 7–17, 2009.

[57] B. Cinaz, B. Arnrich, R. Marca, and G. Tröster, "Monitoring of mental workload levels during an everyday life office-work scenario," *Personal and ubiquitous computing*, vol. 17, no. 2, pp. 229–239, 2013.

[58] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2019. [Online]. Available: https://www.r-project.org/

[59] R. McElreath, *rethinking: Statistical Rethinking book package*, 2019.

[60] H. A. Al-Mutawa, J. Dietrich, S. Marsland, and C. McCartin, "On the shape of circular dependencies in java programs," in *2014 23rd Australian Software Engineering Conference*. IEEE, 2014, pp. 48–57.

[61] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[62] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.

[63] R. Inglehart and C. Welzel, "Changing mass priorities: The link between modernization and democracy," *Perspectives on Politics*, vol. 8, no. 2, pp. 551–567, 2010.

A

Pre-task Manuscript

The manuscript used for the pre-task instructions are shown in this appendix. The instructions introduce the study to the participants and ensures that ethical concerns are addressed.

Pre-task

Good day!

First of all, we'd like to thank you for participating in our study. I'm Jesper and this is Erik.

This study aims to investigate if code maintainability can affect the feelings of software practitioners. Have you previously participated in any studies?

This study is not a test of any kind. There are no right or wrong answers, and we will not judge your performance in any way. And we will definitely not report your answers to your manager or your peers.

We will ask you to perform a couple of programming-related tasks on code examples and then report your feelings with an instrument called SAM, which we will introduce later. Again, there are no right or wrong answers to these tasks. They simply serve to guide your interaction with the code examples and make them more concrete.

Before we start, we'd like you to read and sign a confidentiality agreement. Any data you provide will be anonymized so that it cannot be linked back to you. The only way to link your data to you will be with a single physical list which will be destroyed at the conclusion of our study. Normally, the list will never have to be used, unless

- you happen to think about something that might invalidate your data; or
- if you want us to erase your data; or
- if we need some clarification of your responses.

If you for any reasons want to get into contact with us, please find our email address at the back of the agreement.

Before we start, is there anything we can clarify?

We will now play an audio file which explains how to use the measurement instrument, as well as your task in the experiment. Please, listen.

B

Confidentiality Agreement

This appendix presents the assurance of anonymity provided to the participants. It describes the purpose of the study, how the participant's data would be handled and their rights. The document also allow the participants to fill out whether or not they want to receive a copy of the results and informs them about how to get into contact with the students and their supervisors.

Confidentiality agreement

This experiment is conducted by the M.Sc. students Jesper Olsson and Erik Risfelt of Chalmers University of Technology, Department of Software Engineering. The aim of the experiment is to investigate how technical debt might influence the affective state of software practitioners.

This agreement and any personal information you add to it will not be transferred to any other format, with the exception of your name. Your name will be written along with your anonymous code on a physical list which will be destroyed at the conclusion of the project. This is the only place where your name and anonymous code will be present together. All data and answers you provide as part of the experiment will be marked solely with your anonymous code, and will not be stored with the list of names and anonymous codes. Please do not add any personal information on any other material you provide as part of this experiment.

The experiment will remain anonymous, meaning that participation cannot be traced directly or indirectly back to you. The data is collected only for scientific research purposes in the public interest. The data will be published in raw or aggregated form. Key excerpts and concepts from the experiment may become part of one or more scientific publications, but no personal data will be included.

We would like to emphasise that:

- your participation is entirely voluntary;
- you are free to decline answering any question;
- you are free to withdraw from the study at any time;
- the study will be made publically available;
- at any time before the study is published, you may request your data to be erased.

I have read the above information and by signing this document. I give my consent in participating in the study. I agree not to share any of the contents of the study to any but the parties mentioned herein, until the study is publically available.

Signature: _____

Printed name: _____

Place and date: _____

In the event that the data collected is missing or unusual, the researchers may use the email address below to contact me.

Email address: _____

☐ I would like to receive a copy of the results of the study.

This experiment is part of a master thesis at the software engineering programme at Chalmers University of Technology. Any questions about the study or its contents can be directed to the students in charge.

Jesper Olsson is a master student at the Software Engineering and Technologies programme at Chalmers University of Technology. His studies have focused on large scale systems, software quality and software design and architecture. Contact him at jesploss@student.chalmers.se.

Erik Risfelt is also a master student at the Software Engineering and Technologies programme at Chalmers University of Technology. His studies have focused on software modeling and requirement specification, as well as some functional programming. Contact him at risfelt@student.chalmers.se

The study is supervised by senior researchers. Any questions or complaints concerning the students can be directed to the supervisors.

Terese Besker is a Ph.D. candidate in Software Engineering at Chalmers University of Technology, Sweden. She is working in the research fields of architectural technical debt management. Before becoming a Ph.D. student, she worked as a senior software engineer in the software industry for more than fifteen years. She also has a bachelor degree in software engineering and a master degree in applied IT from Chalmers. She has published several peer-reviewed articles in conference and workshop proceedings. Contact her at besker@chalmers.se

Antonio Martini is an associated professor in Software Engineering at the University of Oslo, Norway. Antonio has worked as a developer in industry, carrying out various roles. Antonio is leading a project within a research consortium of academia and large international companies. Its current focus is on Architectural Technical Debt: a financial metaphor that represents sub-optimal architectural solutions taken as debt that needs to be repaid in the future with extra-effort (interest). Contact him at antonima@ifi.uio.no

C

SAM Instructions

This appendix holds the instructions the participant received in order to understand the measurement instrument and what tasks they were asked to perform during the experiment.

We thank you for coming today and appreciate your participation in this experiment. As we have previously stated, in this study we are interested in how people respond to a few different programming situations that may occur in practice. For about the next 40 minutes, you will be handed software scenarios and you will, for a few minutes, extract the software design and consider its quality. Then, you will be rating each scenario in terms of how it made you feel while working with it. There are no right or wrong answers, so simply respond as honestly as you can.

If you'll look in the rating booklet, you will see 3 sets of 9 figures, each arranged along a scale. We call this set of figures SAM, and you will be using these figures to rate how you felt while working with each scenario. You will use one page – make all 3 ratings – for each scenario that you observe. SAM shows three different kinds of feelings: Happy vs. Unhappy, Excited vs. Calm, and Controlled vs. In-control.

You can see that each SAM figure varies along each scale.

<place 'SAM EXAMPLE RATING: HAPPY VS. UNHAPPY' in front of the participant.>

The first SAM scale is the happy-unhappy scale, which ranges from a smile to a frown. At one extreme of the happy vs. unhappy scale, you felt happy, pleased, satisfied, contented, hopeful. If you felt completely *happy* while working with the scenario, you can indicate this by placing an "X"

<point on the SAM figure, on the first row, that is marked.>

over the figure at the left of the row. The other end of the scale is when you felt completely, unhappy, annoyed, unsatisfied, melancholic, despaired, bored. You can indicate feeling completely *unhappy* by placing an "X"

<point on the SAM figure, on the second row, that is marked.>

on the figure at the right. The figures also allow you to describe intermediate feelings of pleasure, by placing an "X" over any of the other pictures. If you felt completely neutral, neither happy nor sad, place an "X"

<point on the SAM figure, on the third row, that is marked.>

over the figure in the middle. If, in your judgement, your feeling of pleasure or displeasure falls *between* two of the pictures, then place an "X"

<point on the SAM figure, on the fourth row, that is marked.>

between the figures. This permits you to make more finely graded ratings of how you feel in reaction to the scenarios.

<place 'SAM EXAMPLE RATING: EXITED VS. CALM' in front of the participant.>

The excited vs. calm dimension is the second type of feeling displayed here. At one extreme of the scale you felt stimulated, excited, frenzied, jittery, wide-awake, aroused. If you felt completely *aroused* while viewing the picture, place an "X" over the figure at the left. On the other hand, at the other end of the scale, you felt completely relaxed, calm, sluggish, dull, sleepy, unaroused. You can indicate you felt completely *calm* by placing an "X" over the figure at the right. As with the happy-unhappy scale, you can represent intermediate levels by placing an "X" over any of the other figures, or between the figures.

<place 'SAM EXAMPLE RATING: CONTROLLED VS. IN-CONTROL' in front of the participant.>

The last scale of feeling that you will rate is the dimension of controlled vs. in-control. At one end of the scale you have feelings characterised as completely controlled, influenced, cared-for, awed, submissive,

guided. Please indicate feeling *controlled* by placing an "X" over the figure at the left. At the other extreme of this scale, you felt completely controlling, influential, in-control, important, dominant, autonomous. You can indicate that you felt *dominant* by placing an "X" over the figure at the right. Note that the figure is large when you feel important and influential, and that the figure is small when you feel controlled and guided. Remember that you can also represent your feelings between these endpoints. Either place an "X" over any of the intermediate figures, or *between* them.

Some of the scenarios may prompt emotional experiences; others may seem relatively neutral. Your rating of each scenario should reflect your immediate personal experience, and no more. Please rate each one **as you actually felt while you worked with the code example.**

The procedure will be as follows: In the beginning of each scenario, you will be given a code example written in Java. Your task is to communicate potential design issues in the examples, from the perspective of software evolution and maintenance. For example, you might find problems related to understanding the code, extending it with new features or testing its behaviour. **Please write down or speak aloud whatever comes to your mind as you navigate the example.** In order to focus your reasoning, we want you to draw a diagram of the software components and their interrelationships.

When you create the diagram, you do not have to follow any standard (such as UML). It is sufficient that the diagrams are representations of the software at a higher level of abstraction and is clear enough to help you reason about potential problems in the code design. You will have five minutes to work with each example. Some of the examples may be too large to complete within the allotted time. The experiment is designed this way on purpose. Similarly, for some of the examples, you may find yourself feeling finished before the time is up. If this is the case, please keep studying the example, as there may be more aspects to the code design.

When the time is up, we will ask you to fill out SAM. As we've already said, it is very important not to dwell on your ratings. After you've filled out SAM, there will be a short waiting period of two minutes before we start the next scenario. Please, take that time to put aside any emotions induced by the last scenario.

Just as a reminder before we begin; after the time for working with the scenario is up, make your ratings on all 3 dimensions as quickly as possible. It is important that we have information from you on all of these code examples. There are no right or wrong answers; *so rate every scenario on all three dimensions.*

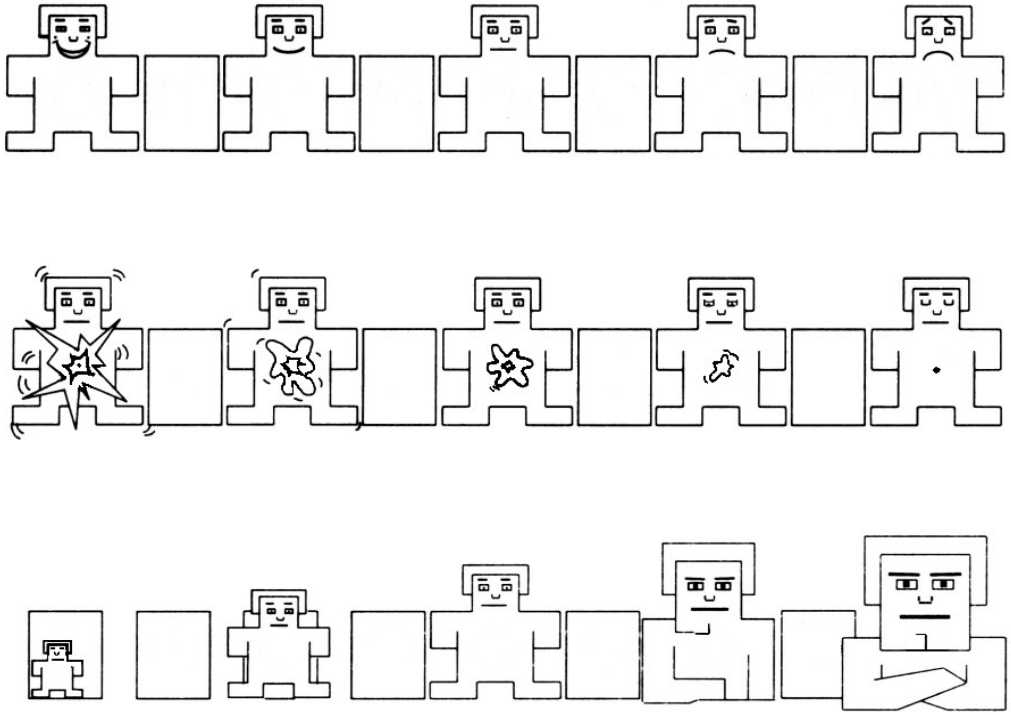
D

SAM Rating Booklet

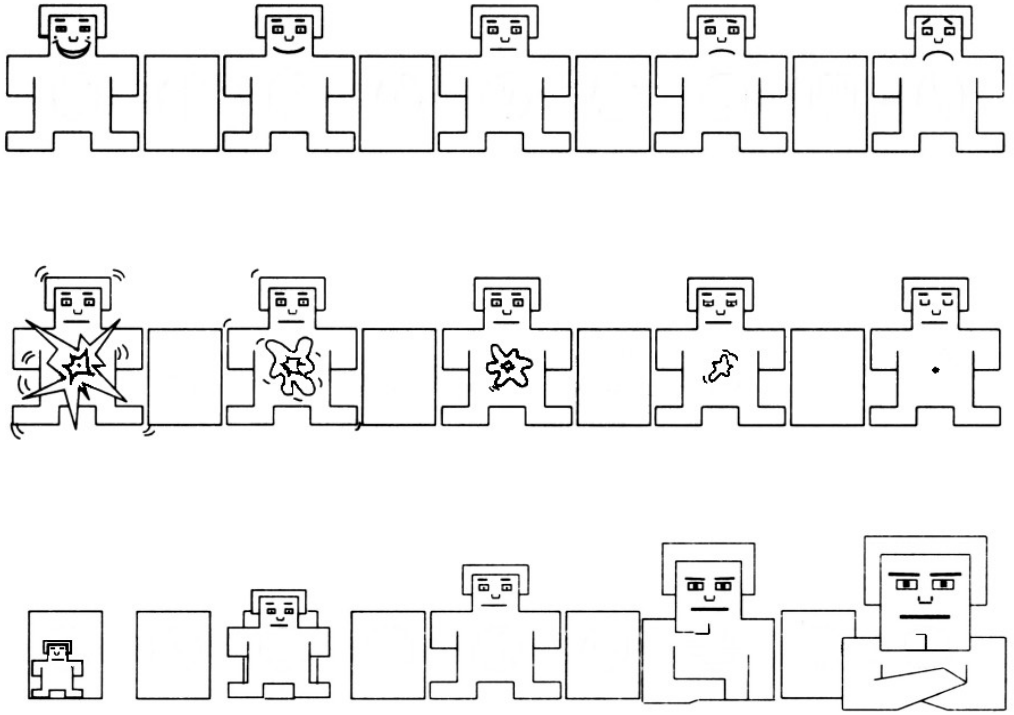
This appendix provides the SAM booklet that was used during the experiment. The booklet is in A5 format and includes, in addition to the cover, six pages of the SAM. The SAM is copyright material and specific permissions have been obtained for use in this study.

SAM

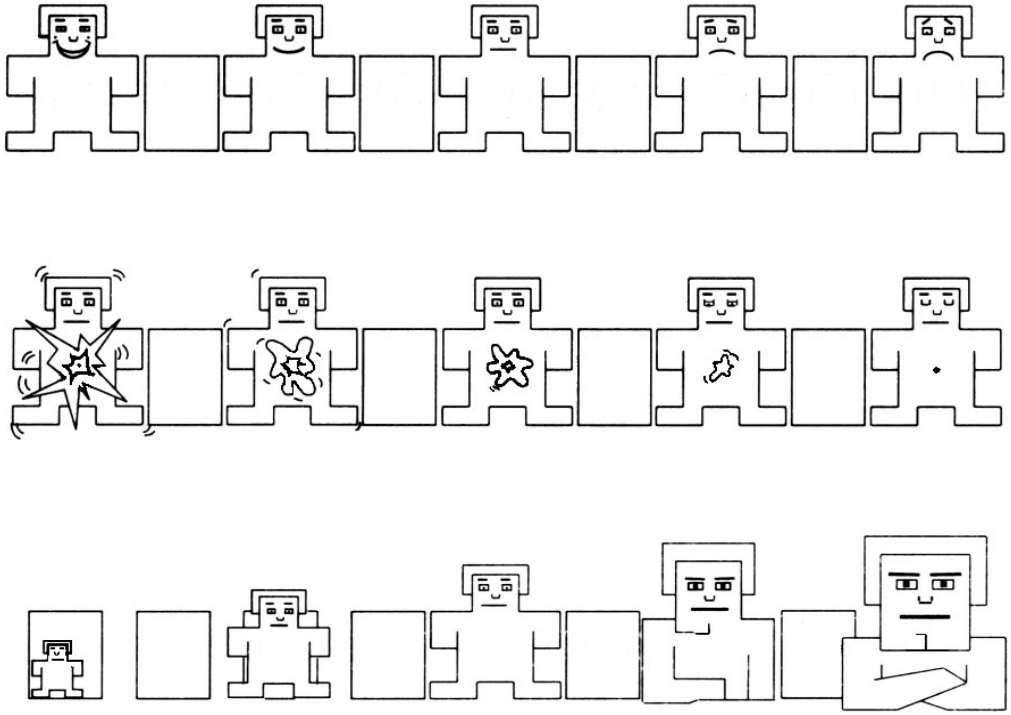
RATING BOOKLET

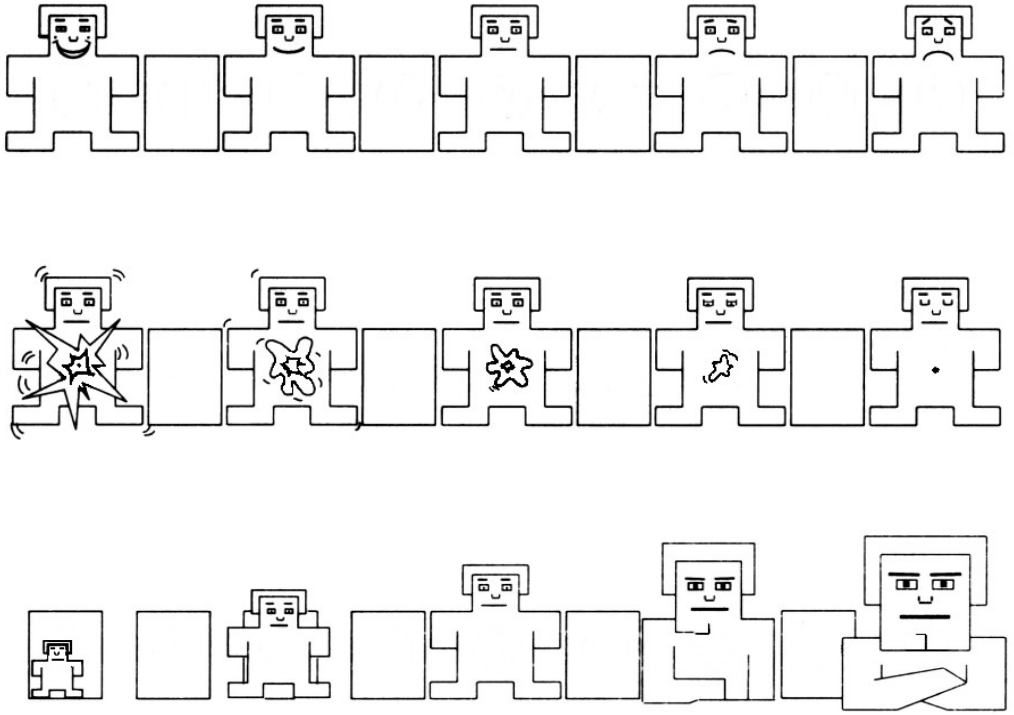


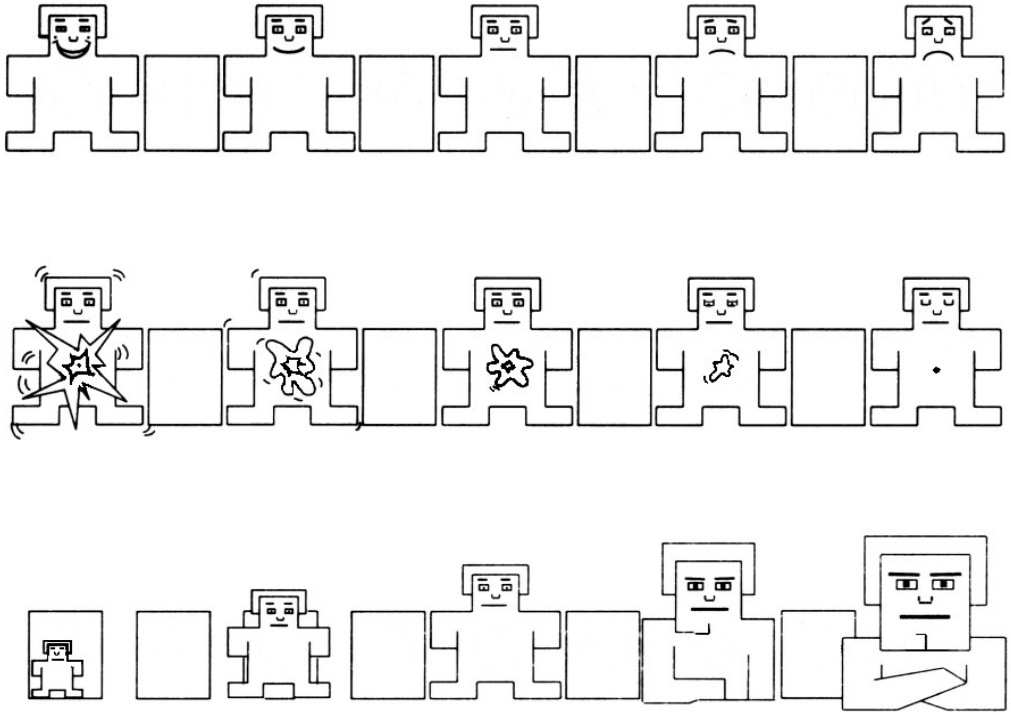
1



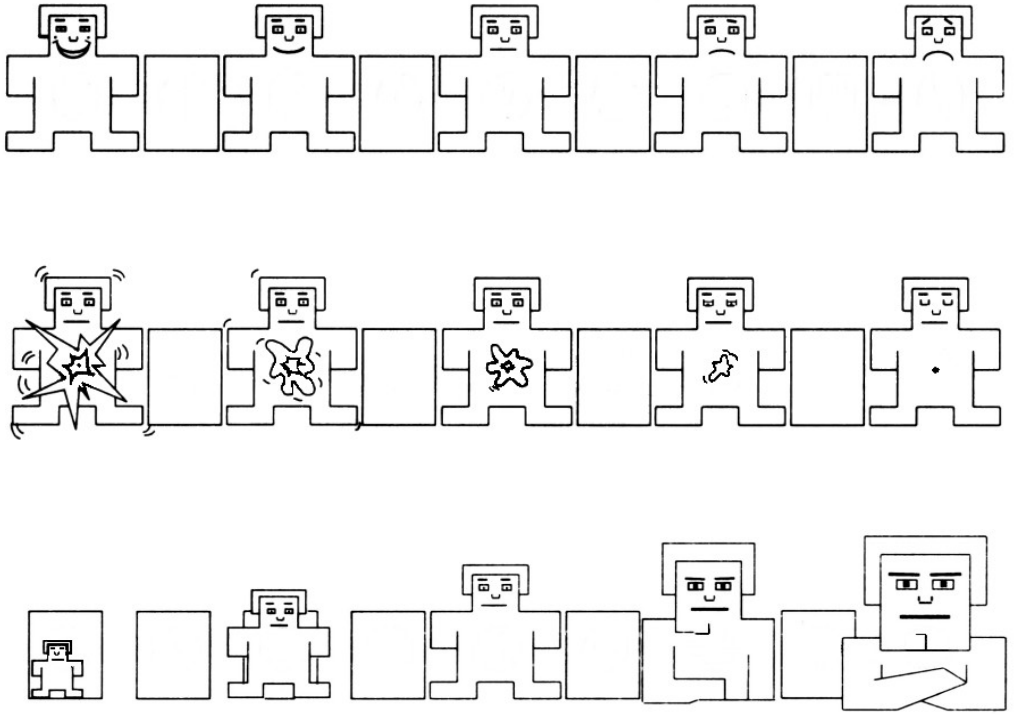
6







5



2

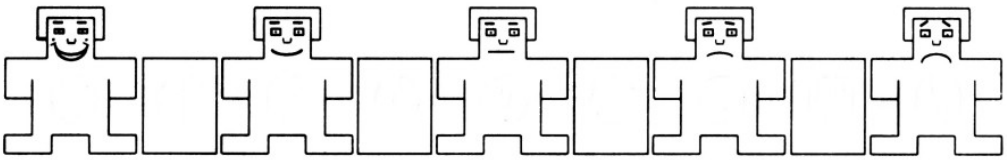
E

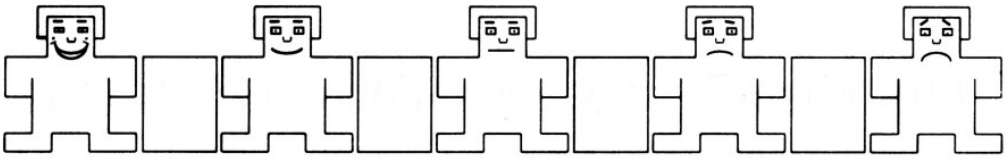
SAM Example Ratings

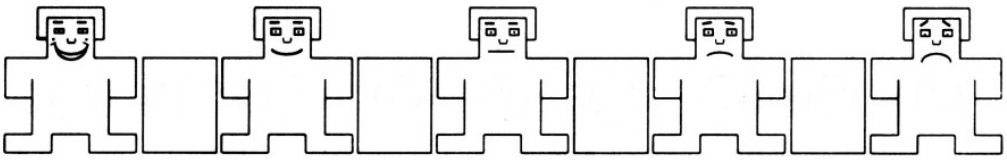
This appendix contains the digital versions of the images used to show the participants how to use the SAM measurement instrument. In the printed version, an "X" was drawn over one figure on each row of every example page, using a regular pen. Each page was then laminated.

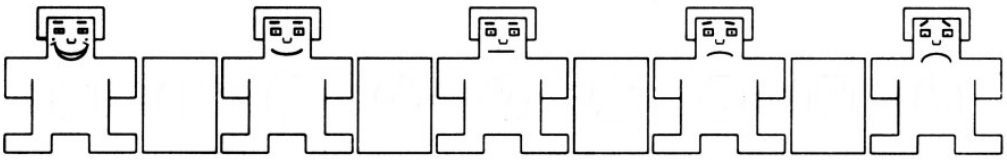
Specifically, on the first row of every page, the "X" was drawn over the left-most (first) figure. On the second row, the "X" was drawn over the right-most (ninth) figure. On the third row, the "X" was drawn over the middle (fifth) figure. Finally, on the last row, the "X" was drawn over the fourth figure on the happy vs. unhappy example, the eighth figure on the excited vs. calm example and the sixth figure on the controlled vs. in-control example.

SAM EXAMPLE RATING: HAPPY VS. UNHAPPY

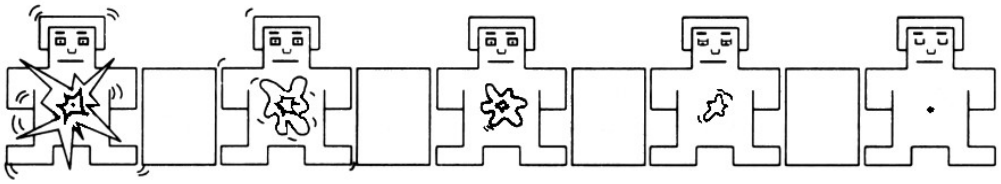


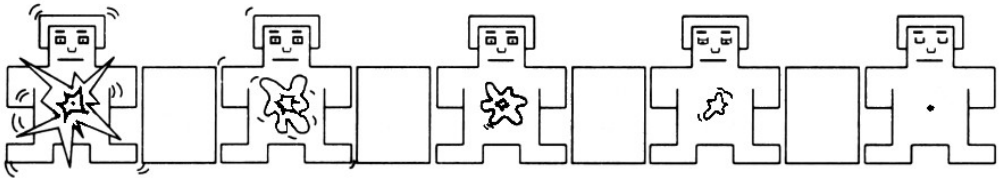


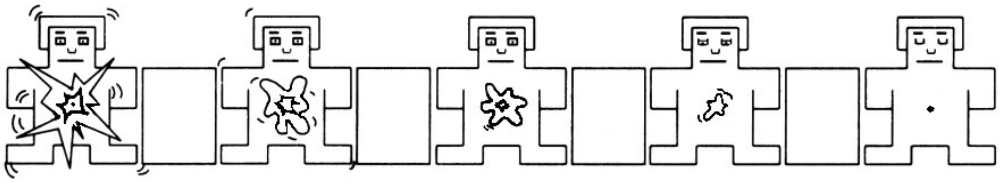


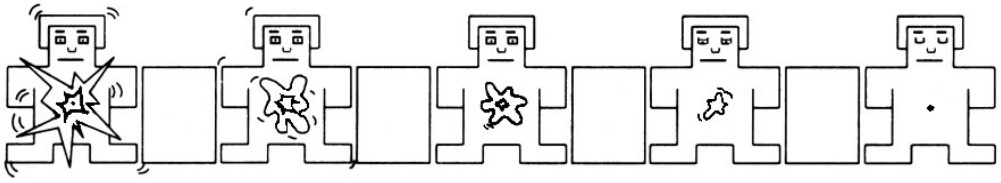


SAM EXAMPLE RATING: EXCITED VS. CALM

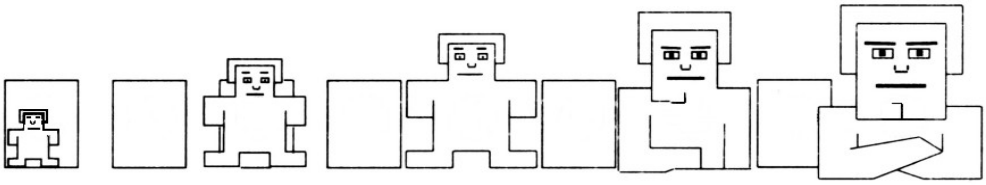


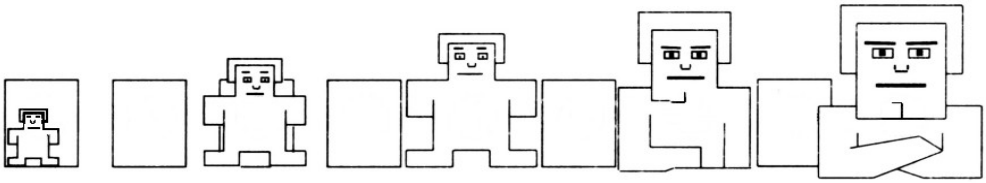


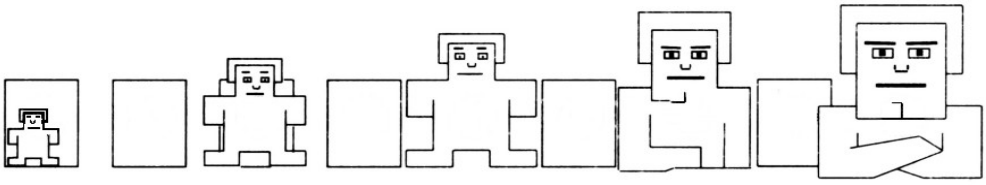


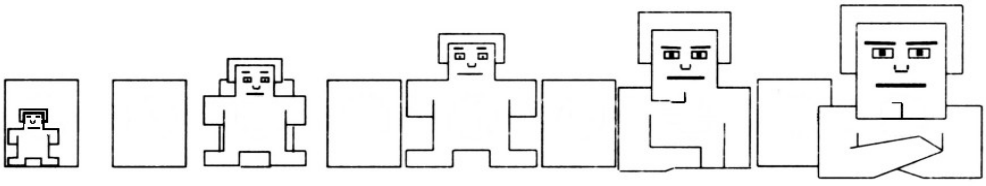


SAM EXAMPLE RATING: CONTROLLED VS. IN-CONTROL









F

Scenarios

This appendix presents the source code for the code examples used in the laboratory experiment. The first example is the anchor point scenario, which exists in only one level. The remaining examples follow sequentially. Each scenario is present first in its low level version, followed by the high level example. This information is summarised in Table F.1. All page numbers listed in the table are inclusive and relative to this page (page number 0). Note that the anchor point only comes in one variant.

Table F.1: Table clarifying which scenario is which in the this appendix.

Page(s)	Scenario	Level
1	Anchor	H
2	A	L
3–4	A	H
5	B	L
6	B	H
7	C	L
8	C	H
9	D	L
10	D	H
11–12	E	L
13–14	E	H

```java
public class Report {
  /* Constructor omitted */
}


public class CopyReport {

  /* Constructor omitted */

  public Report copy(Report report1, Report report2) {
    /* Implementation omitted */
  }
}


public class CreateReport {

  /* Constructor omitted */

  public Report create(Path path) {
    /* Implementation omitted */
  }
}


public class DisplayReport {

  public void display() {
    /* Implementation omitted */
  }
}
```

```java
public abstract class Shape {

  protected Renderer renderer;

  /* Constructor and methods omitted. */

}



public class Ellipse extends Shape {
  /* Implementation omitted. */
}



public class Rectangle extends Shape {
  /* Implementation omitted. */
}



public class Triangle extends Shape {
  /* Implementation omitted. */
}



public interface Renderer {
  /* Implementation omitted. */
}



public class PngRenderer implements Renderer {
  /* Implementation omitted. */
}



public class JpegRenderer implements Renderer {
  /* Implementation omitted. */
}



public class GifRenderer implements Renderer {
  /* Implementation omitted. */
}
```

```
public abstract class Renderer {
  /* Implementation omitted */
}


public class EllipseRenderer extends Renderer {
  /* Implementation omitted. */
}


public class PngEllipseRenderer extends EllipseRenderer {
  /* Implementation omitted. */
}


public class JpegEllipseRenderer extends EllipseRenderer {
  /* Implementation omitted. */
}


public class GifEllipseRenderer extends EllipseRenderer {
  /* Implementation omitted. */
}


public class RectangleRenderer extends Renderer {
  /* Implementation omitted. */
}


public class PngRectangleRenderer extends RectangleRenderer {
  /* Implementation omitted. */
}


public class JpegRectangleRenderer extends RectangleRenderer {
  /* Implementation omitted. */
}


public class GifRectangleRenderer extends RectangleRenderer {
  /* Implementation omitted. */
}


public class TriangleRenderer extends Renderer {
  /* Implementation omitted. */
}
```

```
public class PngTriangleRenderer extends TriangleRenderer {
  /* Implementation omitted. */
}


public class JpegTriangleRenderer extends TriangleRenderer {
  /* Implementation omitted. */
}


public class GifTriangleRenderer extends TriangleRenderer {
  /* Implementation omitted. */
}
```

```
class XpStyle {

  /* Constructor, certain inner classes and certain methods omitted */

  private class XpFillBorder extends LineBorder implements UIResource {

    XpFillBorder(Color color, int thickness) {
      super(color, thickness);
    }

    public Insets getBorderInsets(Component c, Insets insets) {

      Insets margin = null;

      if (c instanceof MarginSupported) {

        margin = ((MarginSupported)c).getMargin();

      }

      insets.top = (margin != null ? margin.top : 0) + thickness;
      insets.left = (margin != null ? margin.left : 0) + thickness;
      insets.bottom = (margin != null ? margin.bottom : 0) + thickness;
      insets.right = (margin != null ? margin.right : 0) + thickness;

      return insets;
    }
  }
}
```

```
class XpStyle {

  /* Constructor, certain inner classes and certain methods omitted */

  private class XpFillBorder extends LineBorder implements UIResource {

    XpFillBorder(Color color, int thickness) {
      super(color, thickness);
    }

    public Insets getBorderInsets(Component c, Insets insets) {

      Insets margin = null;

      if (c instanceof AbstractButton) {

        margin = ((AbstractButton)c).getMargin();

      } else if (c instanceof JToolBar) {

        margin = ((JToolBar)c).getMargin();

      } else if (c instanceof JTextComponent) {

        margin = ((JTextComponent)c).getMargin();

      }

      insets.top = (margin != null ? margin.top : 0) + thickness;
      insets.left = (margin != null ? margin.left : 0) + thickness;
      insets.bottom = (margin != null ? margin.bottom : 0) + thickness;
      insets.right = (margin != null ? margin.right : 0) + thickness;

      return insets;
    }
  }
}
```

```
public class Device {

  public String deviceId;

  public String devicePath;

  public String sourcePath;

  public Boolean enabled;

  public Device proxy;

  /* Constructor, certain members and certain methods omitted. */

  public String getDeviceKeyFromId(String string) {
    /* Implementation omitted. */
  }

  public String lookupDevice(String string) {
    /* Implementation omitted. */
  }

  public String getDevicePath(String string) {
    /* Implementation omitted. */
  }

  public Boolean setDevicePath(String string1, String string2) {
    /* Implementation omitted. */
  }

  public Capabilities getCapabilities(String string) {
    /* Implementation omitted. */
  }

  public Device getProxy(String string) {
    /* Implementation omitted. */
  }

  public Boolean isEnabled() {
    /* Implementation omitted. */
  }

  public Boolean isDecodeDataEnable() {
    /* Implementation omitted. */
  }
}
```

```
public class Device {

  private DeviceData deviceData;

  /* Constructor and certain methods omitted. */

  public String getDeviceKeyFromId(String string) {
    /* Implementation omitted. */
  }

  public String lookupDevice(String string) {
    /* Implementation omitted. */
  }

  public String getDevicePath(String string) {
    /* Implementation omitted. */
  }

  public Boolean setDevicePath(String string1, String string2) {
    /* Implementation omitted. */
  }

  public Capabilities getCapabilities(String string) {
    /* Implementation omitted. */
  }

  public Device getProxy(String string) {
    /* Implementation omitted. */
  }

  public Boolean isEnabled() {
    /* Implementation omitted. */
  }

  public Boolean isDecodeDataEnable() {
    /* Implementation omitted. */
  }
}


public class DeviceData {

  public String deviceId;

  public String devicePath;

  public String sourcePath;

  public Boolean enabled;

  public Device proxy;

  /* Constructor and certain members omitted. */

}
```

```
public class Order {

  private Collection<Item> items;

  /* Constructor and certain methods omitted. */

  public Collection<Item> getItems() {
    return items;
  }

  public double getAmount() {
    return computeAmount();
  }

  private double computeAmount() {
    double amount = 0;

    for (Item item: getItems()) {
      amount += item.getCost();
    }

    return amount;
  }
}


public class Item {

  private double cost;

  /* Constructor and certain methods omitted. */

  public double getCost() {
    return cost;
  }
}
```

```
public class Order {

  private OrderImpl orderImpl;

  private Collection<Item> items;

  /* Constructor and certain methods omitted. */

  public Collection<Item> getItems() {
    return items;
  }

  public double getAmount() {
    return orderImpl.computeAmount();
  }
}


public class OrderImpl {

  private Order order;

  /* Constructor and certain methods omitted. */

  public double computeAmount() {
    double amount = 0;

    for (Item item: order.getItems()) {
      amount += item.getCost();
    }

    return amount;
  }
}


public class Item {

  private double cost;

  /* Constructor and certain methods omitted. */

  public double getCost() {
    return cost;
  }
}
```

```java
public abstract class Document {

  /* Constructor and certain methods omitted. */

  public abstract String read();

  public void save() {
    /* Implementation omitted. */
  }

  public void delete() {
    /* Implementation omitted. */
  }
}


public class ReadOnlyDocument extends Document {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }
}


public abstract class ChangeableDocument extends Document {

  public abstract void write(String content);
}


public class TaggedDocument extends ChangeableDocument {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }

  @Override
  public void write(String content) {
    /* Implementation omitted. */
  }
}
```

```java
public class WritableDocument extends ChangeableDocument {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }

  @Override
  public void write(String content) {
    /* Implementation omitted. */
  }
}
```

```java
public abstract class Document {

  /* Constructor and certain methods omitted. */

  public abstract String read();

  public abstract void write(String content);

  public void save() {
    /* Implementation omitted. */
  }

  public void delete() {
    /* Implementation omitted. */
  }
}


public class ReadOnlyDocument extends Document {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }

  @Override
  public void write(String content) {
    throw new UnsupportedOperationException();
  }
}


public class TaggedDocument extends Document {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }

  @Override
  public void write(String content) {
    /* Implementation omitted. */
  }
}
```

```java
public class WritableDocument extends Document {

  /* Constructor and certain methods omitted. */

  @Override
  public String read() {
    /* Implementation omitted. */
  }

  @Override
  public void write(String content) {
    /* Implementation omitted. */
  }
}
```

G

Post-task Manuscript

This appendix contains the manuscript for the post-task interview of the experiment. Note how the three conceptual parts are marked with horizontal rules. The questionnaire was handed out at the first rule.

Post-task

Thank you for your responses. We have completed the main part of the experiment.

We will now ask you to fill out a questionnaire about your professional experience with software. Please avoid sharing information that can be used to uniquely identify you.

Thank you. We have now reached the final part of the study, during which we will ask you some more open questions about how you perceived the study and your perception of code maintainability and feelings in general.

All your opinions on the subject are of interest to us, so please speak freely about whatever comes to your mind.

May we use audio recording in order to accurately transcribe your answers? The record will only be played back to the two of us.

IQ1.1: Could you please tell us more about your daily work. What type of tasks do you normally encounter?
IQ1.2: How do those tasks make you feel?
IQ1.3: Do you face challenges in those tasks?
IQ1.4: How do those challenges make you feel?
IQ1.5: Are those feelings frequent?

IQ2: In contrast to challenging tasks, what sorts of feelings would you say you get from routine tasks?

IQ3: Do you think that anything outside of this experiment did impact your responses today?

IQ4.1: Would you please tell us about how you experienced the code examples?
IQ4.2: What about the software design in the examples?

IQ5: What would you say are the differences between the scenarios we provided and software one encounters in industry?

IQ6: Did you find SAM difficult to use or understand?

IQ7: That was all of the questions that we had for you. Is there anything you would like to add?

If you come up with any questions at a later time, please feel free to email us. We would like to thank you one final time for participating in our study and remind you that it is important that you do not discuss this experiment with anyone until the end of April.

H

Participant Professional Profile Questionnaire

Included in this appendix is the questionnaire used to collect data about the professional background of the participants.

Participant profiling questionnaire

This questionnaire aims to extract general information about the participants' professional experience. This data serves to investigate whether there is any significant variation in regards to different characteristics of the respondents. You have the option to not answer specific questions by simply leaving it blank.

Q1: My highest level of completed academic education is

- ☐ None
- ☐ Some bachelor studies
- ☐ Bachelor degree
- ☐ Some master studies
- ☐ Master degree
- ☐ Some Ph. D. studies
- ☐ Ph. D.
- ☐ Other: _____

Q2: My education major (e.g. computer science, electrical engineering, software engineering) was

Q3: I have working experience with software for _____ years.

Q4: My current role (e.g. architect, developer, tester, ...) is

Q5: The programming language I am most experienced in is

Please list only one

Q6: My currently preferred programming language is

Please list only one

Q7: Most of my working experience comes from the following domain (e.g. telecom, healthcare, finance, ...)

Do you have any additional comments concerning this questionnaire?

I

Data Set

This appendix contains the data set that was used as input for the statistical model. Not only does this transparency strengthen the validity of the study, but also enables the data to be used for inferences in other research.

The data set is presented in Table I.1. It contains five repeated measures for how five different software scenarios, existing in two levels of ATD, impacted the affects of 40 software practitioners. The table uses the long format.

Table I.1: The data set used for statistical inference in this study.

ID	EX	ENTITIES	V	A	D	EDU	MAJOR	EXP	ROLE	LANG
1	DL	3	7	6	7	5	CS	17	D	J
1	EH	4	9	4	4	5	CS	17	D	J
1	AH	13	5	5	5	5	CS	17	D	J
1	CL	2	7	7	3	5	CS	17	D	J
1	BH	10	9	5	1	5	CS	17	D	J
2	AH	13	9	7	7	5	CS	12	A	P
2	DL	3	3	8	7	5	CS	12	A	P
2	EL	5	2	3	9	5	CS	12	A	P
2	BH	10	9	1	9	5	CS	12	A	P
2	CL	2	5	5	7	5	CS	12	A	P
3	DH	4	8	3	2	3	CS	20	D	J
3	BL	8	2	8	9	3	CS	20	D	J
3	EL	5	5	7	7	3	CS	20	D	J
3	AH	13	3	8	7	3	CS	20	D	J
3	CL	2	8	3	2	3	CS	20	D	J
4	CH	3	4	5	7	3	CS	2	D	J
4	AL	8	4	6	8	3	CS	2	D	J
4	EH	4	6	7	7	3	CS	2	D	J
4	DL	3	3	8	8	3	CS	2	D	J
4	BH	10	4	7	5	3	CS	2	D	J
5	AH	13	7	3	5	4	SE	10	A	CS
5	CL	2	8	6	7	4	SE	10	A	CS
5	DL	3	1	5	2	4	SE	10	A	CS
5	BH	10	8	4	7	4	SE	10	A	CS
5	EL	5	7	3	6	4	SE	10	A	CS
6	AH	13	3	7	6	3	CS	20	D	A
6	CL	2	7	4	3	3	CS	20	D	A

6	EL	5	7	3	2	3	CS	20	D	A
6	DH	4	7	6	2	3	CS	20	D	A
6	BL	8	8	3	2	3	CS	20	D	A
7	EL	5	6	6	2	3	SE	2	T	CS
7	DH	4	4	3	3	3	SE	2	T	CS
7	CH	3	3	6	5	3	SE	2	T	CS
7	AL	8	6	6	2	3	SE	2	T	CS
7	BH	10	5	5	5	3	SE	2	T	CS
8	EH	4	4	6	6	5	SE	1	D	CS
8	CL	2	5	7	5	5	SE	1	D	CS
8	AL	8	4	6	4	5	SE	1	D	CS
8	DH	4	6	6	3	5	SE	1	D	CS
8	BL	8	6	7	2	5	SE	1	D	CS
9	EL	5	1	1	9	5	SE	11	D	SAP
9	DH	4	1	1	7	5	SE	11	D	SAP
9	CH	3	1	1	7	5	SE	11	D	SAP
9	BL	8	3	1	7	5	SE	11	D	SAP
9	AH	13	1	5	9	5	SE	11	D	SAP
10	BL	8	6	6	3	3	SE	14	D	JS
10	EH	4	4	5	6	3	SE	14	D	JS
10	DH	4	5	4	3	3	SE	14	D	JS
10	CL	2	6	5	7	3	SE	14	D	JS
10	AH	13	3	5	3	3	SE	14	D	JS
11	BL	8	7	3	4	3	SE	9	A	CS
11	AH	13	7	5	6	3	SE	9	A	CS
11	CH	3	3	5	7	3	SE	9	A	CS
11	EL	5	5	3	8	3	SE	9	A	CS
11	DH	4	2	3	9	3	SE	9	A	CS
12	BH	10	3	5	5	1	NONE	3	D	CS
12	DL	3	3	3	3	1	NONE	3	D	CS
12	CL	2	3	3	5	1	NONE	3	D	CS
12	AH	13	3	3	6	1	NONE	3	D	CS
12	EL	5	3	7	5	1	NONE	3	D	CS
13	CH	3	7	6	3	3	CS	9	M	CS
13	AL	8	3	8	8	3	CS	9	M	CS
13	DL	3	5	8	3	3	CS	9	M	CS
13	BH	10	9	3	5	3	CS	9	M	CS
13	EL	5	3	3	7	3	CS	9	M	CS
14	DL	3	3	5	3	2	NONE	20	D	CS
14	EH	4	5	4	4	2	NONE	20	D	CS
14	BH	10	7	7	5	2	NONE	20	D	CS
14	AL	8	3	5	6	2	NONE	20	D	CS
14	CH	3	4	4	7	2	NONE	20	D	CS
15	BL	8	6	3	2	5	SE	20	A	R
15	EH	4	7	4	4	5	SE	20	A	R
15	CH	3	7	3	2	5	SE	20	A	R

15	DL	3	4	6	7	5	SE	20	A	R
15	AH	13	9	3	5	5	SE	20	A	R
16	BL	8	3	3	2	3	SE	6	D	CS
16	EH	4	3	7	7	3	SE	6	D	CS
16	AH	13	2	6	4	3	SE	6	D	CS
16	DL	3	3	4	7	3	SE	6	D	CS
16	CH	3	4	5	3	3	SE	6	D	CS
17	AL	8	4	7	2	3	SE	8	D	CS
17	BH	10	6	4	5	3	SE	8	D	CS
17	CH	3	5	3	3	3	SE	8	D	CS
17	DL	3	2	5	8	3	SE	8	D	CS
17	EH	4	3	3	8	3	SE	8	D	CS
18	BH	10	7	7	4	3	SE	4	SD	CS
18	DL	3	2	3	8	3	SE	4	SD	CS
18	CL	2	5	6	6	3	SE	4	SD	CS
18	AH	13	6	5	5	3	SE	4	SD	CS
18	EL	5	4	6	6	3	SE	4	SD	CS
19	CL	2	5	9	6	5	CS	6	D	P
19	BH	10	5	9	5	5	CS	6	D	P
19	AH	13	5	9	5	5	CS	6	D	P
19	EL	5	5	9	5	5	CS	6	D	P
19	DH	4	5	8	5	5	CS	6	D	P
20	DL	3	3	6	7	7	EE	4	M	CP
20	EH	4	3	5	8	7	EE	4	M	CP
20	CH	3	7	3	5	7	EE	4	M	CP
20	AL	8	7	3	2	7	EE	4	M	CP
20	BH	10	4	3	3	7	EE	4	M	CP
21	AH	13	5	2	9	5	SE	2	D	CS
21	DL	3	4	7	1	5	SE	2	D	CS
21	CL	2	5	6	2	5	SE	2	D	CS
21	EH	4	2	4	5	5	SE	2	D	CS
21	BL	8	6	6	2	5	SE	2	D	CS
22	CH	3	3	5	7	3	BA	2	D	M
22	EL	5	7	7	3	3	BA	2	D	M
22	BL	8	4	8	5	3	BA	2	D	M
22	DH	4	3	9	8	3	BA	2	D	M
22	AL	8	5	3	1	3	BA	2	D	M
23	DL	3	4	5	6	3	EE	18	D	CS
23	EH	4	4	7	7	3	EE	18	D	CS
23	BH	10	6	4	3	3	EE	18	D	CS
23	AL	8	7	5	3	3	EE	18	D	CS
23	CH	3	3	7	7	3	EE	18	D	CS
24	BL	8	5	7	7	5	CS	35	A	CS
24	EH	4	7	5	3	5	CS	35	A	CS
24	AH	13	5	9	9	5	CS	35	A	CS
24	CL	2	7	3	5	5	CS	35	A	CS

24	DH	4	5	7	7	5	CS	35	A	CS
25	DL	3	3	5	5	3	SE	6	D	J
25	CH	3	3	3	7	3	SE	6	D	J
25	BH	10	7	5	9	3	SE	6	D	J
25	AL	8	5	3	5	3	SE	6	D	J
25	EH	4	7	5	7	3	SE	6	D	J
26	DL	3	3	4	8	5	CS	11	A	J
26	CH	3	5	7	8	5	CS	11	A	J
26	AH	13	7	3	3	5	CS	11	A	J
26	EL	5	5	7	8	5	CS	11	A	J
26	BH	10	7	5	5	5	CS	11	A	J
27	DH	4	3	7	7	3	SE	1	D	JS
27	EL	5	9	5	3	3	SE	1	D	JS
27	BL	8	7	5	3	3	SE	1	D	JS
27	AH	13	5	9	9	3	SE	1	D	JS
27	CL	2	1	5	5	3	SE	1	D	JS
28	CH	3	7	4	7	5	CS	4	M	C
28	BL	8	5	7	3	5	CS	4	M	C
28	EL	5	5	7	7	5	CS	4	M	C
28	DH	4	5	9	7	5	CS	4	M	C
28	AL	8	5	7	2	5	CS	4	M	C
29	EH	4	7	5	6	2	SE	15	D	CS
29	AL	8	2	4	7	2	SE	15	D	CS
29	BL	8	7	4	2	2	SE	15	D	CS
29	DH	4	7	7	3	2	SE	15	D	CS
29	CL	2	8	3	3	2	SE	15	D	CS
30	CH	3	5	5	4	2	CS	8	D	CS
30	AL	8	3	5	5	2	CS	8	D	CS
30	EL	5	5	5	5	2	CS	8	D	CS
30	DH	4	4	5	6	2	CS	8	D	CS
30	BL	8	6	6	5	2	CS	8	D	CS
31	AL	8	3	1	1	4	CS	21	A	A
31	CH	3	3	7	1	4	CS	21	A	A
31	DH	4	7	1	1	4	CS	21	A	A
31	BL	8	9	3	1	4	CS	21	A	A
31	EH	4	1	3	7	4	CS	21	A	A
32	AH	13	3	5	6	5	SE	3	M	CS
32	EL	5	7	4	5	5	SE	3	M	CS
32	BL	8	8	3	2	5	SE	3	M	CS
32	DH	4	4	7	6	5	SE	3	M	CS
32	CL	2	7	5	5	5	SE	3	M	CS
33	BH	10	6	6	3	5	SE	2	D	J
33	EL	5	3	7	6	5	SE	2	D	J
33	DL	3	3	7	5	5	SE	2	D	J
33	CH	3	4	7	6	5	SE	2	D	J
33	AL	8	7	7	4	5	SE	2	D	J

34	CH	3		7	7	5	5	CS	6	D	J
34	AL	8		1	5	9	5	CS	6	D	J
34	DL	3		1	5	9	5	CS	6	D	J
34	EH	4		1	3	7	5	CS	6	D	J
34	BL	8		7	5	5	5	CS	6	D	J
35	BH	10		7	3	3	3	SE	6	D	J
35	EL	5		3	8	3	3	SE	6	D	J
35	AL	8		1	4	7	3	SE	6	D	J
35	CH	3		7	7	5	3	SE	6	D	J
35	DL	3		1	4	9	3	SE	6	D	J
36	EL	5		1	3	9	5	CS	19	D	CS
36	BH	10		3	3	7	5	CS	19	D	CS
36	CH	3		5	7	5	5	CS	19	D	CS
36	AL	8		1	3	8	5	CS	19	D	CS
36	DH	4		5	5	3	5	CS	19	D	CS
37	DH	4		8	3	3	4	EP	12	D	CS
37	AL	8		4	7	7	4	EP	12	D	CS
37	CL	2		8	4	2	4	EP	12	D	CS
37	BH	10		6	5	3	4	EP	12	D	CS
37	EL	5		6	6	5	4	EP	12	D	CS
38	EH	4		2	7	8	3	EE	23	SD	J
38	DL	3		7	9	9	3	EE	23	SD	J
38	CL	2		8	7	4	3	EE	23	SD	J
38	BH	10		7	8	3	3	EE	23	SD	J
38	AL	8		5	8	5	3	EE	23	SD	J
39	BL	8		4	4	5	3	CS	4	D	CS
39	CH	3		3	4	7	3	CS	4	D	CS
39	DH	4		6	5	1	3	CS	4	D	CS
39	EL	5		3	3	7	3	CS	4	D	CS
39	AH	13		3	5	8	3	CS	4	D	CS
40	CL	2		8	2	2	5	SE	7	A	CS
40	DH	4		3	8	8	5	SE	7	A	CS
40	BH	10		9	5	1	5	SE	7	A	CS
40	EL	5		8	2	3	5	SE	7	A	CS
40	AH	13		5	7	4	5	SE	7	A	CS

J

Pilot Study Outcome

In this appendix, the outcome of the pilot studies is presented. The first section lists the results of the SAM ratings on the same format that was used in the main study. The second section describes the data collected concerning the study design.

In total, eight students participated in the pilot studies. The first six were each assigned to one of four sessions that was exploratory. The remaining two students attended one session each in confirmatory pilot studies. This information is summarised in Table J.1. Please note that the session numbers are not unique between study types, viz. although participants 105 and 106 took part in the same session, this was not the same session as 107 attended.

Table J.1: The assignment of the participants to the different pilot studies.

ID	Type	Session
105	Exploratory	1
106	Exploratory	1
102	Exploratory	2
104	Exploratory	3
101	Exploratory	4
103	Exploratory	4
107	Confirmatory	1
108	Confirmatory	2

J.1 SAM Ratings

The SAM ratings collected from the participants are shown in Table J.2 and is represented in the long format. It should be noted that the participants who attended the exploratory pilot studies were subjected to scenario ScF, which was later removed. That scenario corresponded to the second example of the Imperative Abstraction design smell in [23]. Similarly, scenario ScR corresponds to the anchor point scenario.

Table J.2: The quantitative data collected during the pilot studies.

ID	EX	ENTITIES	V	A	D	EDU	MAJOR	EXP	ROLE	LANG
101	RH	5	7	3	3	4	SE	0	NONE	J
101	AH	13	3	7	5	4	SE	0	NONE	J
101	BL	8	6	7	4	4	SE	0	NONE	J
101	DH	4	5	8	6	4	SE	0	NONE	J
101	FL	2	7	4	4	4	SE	0	NONE	J
102	FH	5	5	9	3	2	SE	2	D	P
102	RL	2	7	7	2	2	SE	2	D	P
102	BH	10	9	9	2	2	SE	2	D	P
102	AH	13	6	6	1	2	SE	2	D	P
103	RH	5	7	7	5	4	SE	1	M	J
103	BH	10	7	5	3	4	SE	1	M	J
103	DL	3	5	9	4	4	SE	1	M	J
103	FH	5	4	5	3	4	SE	1	M	J
103	AL	8	7	9	5	4	SE	1	M	J
104	RH	5	9	3	5	4	SE	0	D	J
104	AL	8	3	7	7	4	SE	0	D	J
104	FL	2	8	5	3	4	SE	0	D	J
104	BH	10	9	1	5	4	SE	0	D	J
105	BH	10	6	6	6	2	SE	0	NONE	J
105	FL	2	6	4	3	2	SE	0	NONE	J
105	AH	13	8	4	2	2	SE	0	NONE	J
106	BH	10	5	4	3	2	SE	1	NONE	J
106	AL	8	3	6	7	2	SE	1	NONE	J
106	RH	5	7	3	6	2	SE	1	NONE	J
107	RH	5	8	7	2	4	SE	1	D	JS
107	DL	3	3	3	7	4	SE	1	D	JS
107	CH	3	6	5	6	4	SE	1	D	JS
107	BH	10	6	4	4	4	SE	1	D	JS
107	AL	8	2	3	9	4	SE	1	D	JS
107	EH	4	3	4	8	4	SE	1	D	JS
108	RH	5	3	4	4	4	SE	2	D	P
108	DH	4	7	7	4	4	SE	2	D	P
108	CL	2	7	7	3	4	SE	2	D	P
108	BL	8	3	3	7	4	SE	2	D	P
108	AH	13	7	5	7	4	SE	2	D	P
108	EL	5	3	8	7	4	SE	2	D	P

J.2 Study Design Data

Table J.3 details, for each participant in the exploratory pilot studies, how much time the different parts of the study required. The same information is shown in Table J.4 for the confirmatory pilot studies, albeit on a higher level of abstraction.

Table J.3: Time profiles for the participants in the exploratory pilot study.

ID	Item	Time item	Time cumulative (per participant)
105	Pre-task instructions	1:25	1:25
	Confidentiality read & filled	2:35	4:00
	Start/introduce SAM instruction recording	0:51	4:51
	SAM instructions	7:43	12:34
	Scenario 1: ScB-H	5:26	18:00
	Deacclimatisation, task clarifications	1:51	19:51
	Scenario 2: ScF-L	7:00	26:52
	Deacclimatisation	1:27	28:20
	Scenario 3: ScA-H	5:38	33:58
	Post-task instructions	1:26	35:25
	Post-task questions	12:13	47:38
	Thank you's and finish	0:20	48:00
106	Pre-task instructions	1:25	1:25
	Confidentiality read & filled	2:35	4:00
	Start/introduce SAM instruction recording	0:51	4:51
	SAM instructions	7:43	12:34
	Scenario 1: ScB-H	5:26	18:00
	Deacclimatisation, task clarifications	1:51	19:51
	Scenario 2: ScA-L	7:00	26:52
	Deacclimatisation	1:27	28:20
	Scenario 3: ScR-H	5:38	33:58
	Post-task instructions	1:26	35:25
	Post-task questions	12:13	47:38
	Thank you's and finish	0:20	48:00
107	Pre-task instructions	-	-
	Confidentiality read & filled	3:13	3:13
	SAM instructions	7:57	11:10
	Scenario 1: ScF-H	5:38	16:48
	Deacclimatisation, task clarifications	1:01	17:50
	Scenario 2: ScR-L	4:05	21:55
	Deacclimatisation	0:14	22:10
	Scenario 3: ScB-H	5:10	27:21
	Deacclimatisation	0:36	27:57
	Scenario 4: ScA-H	5:03	33:00
	Post-task instructions	0:20	48:00
	Post-task questions	7:07	41:10

	Thank you's and finish	0:18	41:28
104	Pre-task instructions & C.A.	4:31	4:31
	SAM instructions	7:43	12:14
	Scenario 1: ScR-H	3:51	16:06
	Deacclimatisation, task clarifications	1:19	17:25
	Scenario 2: ScA-L	2:48	20:13
	Deacclimatisation	0:54	21:08
	Scenario 3: ScF-L	2:51	24:00
	Deacclimatisation	0:33	24:33
	Scenario 4: ScB-H	3:50	28:23
	Post-task instructions	1:04	29:28
	Post-task questions	6:33	36:02
101	Pre-task instructions & C.A.	4:40	4:40
	SAM instructions	7:55	12:35
	Scenario 1: ScR-H	5:12	17:48
	Deacclimatisation, task clarifications	1:34	19:22
	Scenario 2: ScA-H	5:13	24:36
	Deacclimatisation	1:27	26:03
	Scenario 3 ScB-L	5:13	31:16
	Deacclimatisation	1:27	32:43
	Scenario 4: ScD-H	5:10	37:55
	Deacclimatisation	1:21	39:15
	Scenario 5: ScF-L	5:03	44:19
	Post-task instructions	1:47	46:06
	Post-task questions	21:46	1:07:52
103	Pre-task instructions & C.A.	4:40	4:40
	SAM instructions	7:55	12:35
	Scenario 1: ScR-H	5:12	17:48
	Deacclimatisation, task clarifications	1:34	19:22
	Scenario 2: ScB-H	5:13	24:36
	Deacclimatisation	1:27	26:03
	Scenario 3: ScD-L	5:13	31:16
	Deacclimatisation	1:27	32:43
	Scenario 4: ScF-H	5:10	37:55
	Deacclimatisation	1:21	39:15
	Scenario 5: ScA-L	5:03	44:19
	Post-task instructions	1:47	46:06
	Post-task questions	21:46	1:07:52
107	Pre-task instructions	12:37	12:37
	Measurement sitting	43:19	55:56
	Profile questionnaire	2:05	58:01
	Interview	7:20	1:05:22
108	Pre-task instructions	12:58	12:58
	Measurement sitting	42:56	55:54
	Profile questionnaire	03:52	59:47
	Interview	17:08	1:16:56

Table J.4: Time profiles for the participants in the confirmatory pilot study.

ID	Item	Time item	Time cumulative (per participant)
107	Pre-task instructions	12:37	12:37
	Measurement sitting	43:19	55:56
	Profile questionnaire	2:05	58:01
	Interview	7:20	1:05:22
108	Pre-task instructions	12:58	12:58
	Measurement sitting	42:56	55:54
	Profile questionnaire	03:52	59:47
	Interview	17:08	1:16:56