Analyzing Comprehension of Models of Variable Software Systems with Eye-Tracking Technologies

by

Elmira Rezaei Sepasi

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.Sc.

MONTREAL, AUGUST 09, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

My time as a graduate student has been full of joy and good memories, and it was the best time to accumulate my most invaluable possessions. Sharing this journey with those who I love makes it even more meaningful, and for such a wonderful experience, I owe many thanks.

Words cannot express my gratitude to my supervisor, Prof. Roberto Erick Lopez Herrejon, for his invaluable supports and feed backs. Thank you for believing in me, and for allowing me to experience and learn under your guidance. Without your support, I am sure this path would have been very challenging. Thank you for your all constructive advice and helpful supports.

I would like to thank Prof. Vincent Levesque for his great help during my experiments. I should also thank the librarians, Vanessa Ayotte, technicienne en documentation and Lala Prasun, Agent de recherche for helping me to document my thesis in ETS University dataverse. I am deeply grateful to Erika Olivaux, a Decanat de la recherche on the ethical committee, for her help and support during my study.

In particular, I would like to thank Prof. Sylvie Ratte and Prof. Julien Mercier who accepted to be my jury members.

Morad, my friend, my partner, I thank you for your guidance, patience, and encouragements that always helped me to make the right decisions in challenging situations. This endeavor would not be possible without your support.

My dear parents, Rafat and Saeid, I know how sincerely your hearts wanted me to continue my study. You have always believed in me and filled me with your unending love. Thank you.

My two adorable brothers, Massoud and Arsalan, thank you for sharing your unfiltered thoughts with me every moment. And I am so thankful to have you.

My dear Mona, and my lovely nephew/niece, I am so blessed with having you both. I appreciate all the joy you bring to my life and studies.

My big brother Arash, even if you are not with me now here on the earth, you are always in my heart, and I always keep my promise to you.

I would like to extend my sincere thanks to my colleagues, especially to Kambiz for all his help with the project.

My lovely friend Mastaneh, thank you for all your beautiful smiles, kindness, and great emotional support.

Finally, I would like to acknowledge and thank the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) Foundation (RGPIN-2017-05421, for the project "Mining Software Repositories and Information Visualization for Empirically Robust Testing of Variable Software").

Analyse de la compréhension des modèles de logiciels variables Systèmes avec technologies de suivi des yeux

Elmira Rezaei Sepasi

RÉSUMÉ

Les lignes de produits logiciels (SPL) sont des familles de systèmes logiciels connexes qui se distinguent par l'ensemble des fonctionnalités offertes par chaque système. Les modèles de caractéristiques sont des composants essentiels des lignes de produits logiciels et la norme de facto pour modéliser la variabilité des SPL, car ils décrivent les fonctionnalités, leurs relations et toutes les combinaisons de fonctionnalités qui constituent une SPL. Par conséquent, leur compréhension correcte est cruciale pour l'exécution adéquate de toutes les tâches dans lesquelles elles sont impliquées. À notre connaissance, aucune recherche n'a été menée sur la compréhension des modèles de caractéristiques. Pour combler ce manque, nous proposons une étude empirique de la compréhension des modèles de caractéristiques dans des tâches simples de validation de configuration. Nous proposons un premier modèle cognitif pour ce type de tâches que nous analysons en mesurant les fixations du regard sur les différents éléments visuels impliqués dans les tâches. Nos résultats ont permis d'identifier trois composantes principales du modèle cognitif et leur distribution en termes d'effort cognitif pour la réalisation de ces tâches. Nous soutenons que des recherches plus poussées sur la compréhension des modèles de caractéristiques peuvent éclairer la conception du langage et le développement d'outils afin de fournir des structures de langage, des interfaces utilisateur et un support plus adaptés à ce type de modèles.

Mots-clés: modèles de caractéristiques, lignes de produits logiciels, analyse du regard, eye-trackers

Analyzing Comprehension of Models of Variable Software Systems with Eye-Tracking Technologies

Elmira Rezaei Sepasi

ABSTRACT

Software Product Lines (SPLs) are families of related software systems which are distinguished by the set of features each system provides. Feature Models are pivotal components of Software Product Lines and the de facto standard for modelling the variability of SPLs because they describe the features, their relations, and all the combinations of features that constitute a SPL. Therefore, their correct comprehension is crucial for performing adequately all the tasks where they are involved. According to our knowledge, no research has been conducted on feature model comprehension. To address this lack, we contribute an empirical study of feature model comprehension in simple configuration validation tasks. We propose a first cognitive model for this type of tasks that we analyze by measuring eye gaze fixations on the different visual elements involved in the tasks. Our results identified three main components of the cognitive model and their distribution in terms of the cognitive effort for performing these tasks. We argue that further research on feature model comprehension can inform language design and tool development to provide more suitable language structures, user interfaces and support for this kind of models.

Keywords: feature model, software product lines, gaze analysis, eye-trackers

TABLE OF CONTENTS

Page

INTRODUCTION 1

CHAPTER 1 BACKGROUND 3
1.1 Software Product Lines 3
1.1.1 The goal of Software Product Line 4
1.1.2 Variability modelling approaches 5
1.1.3 Introducing Feature Models 6
1.1.4 Running Example Description 9
1.2 Full and Partial Configurations — Definitions and Examples 11
1.3 Eye-tracking Basics 14
1.3.1 What is eye tracking? 14
1.3.2 What are the different kinds of eye tracker? 15
1.3.3 How does it work? 17
1.3.4 What factors can influence data quality ? 18
1.3.5 How to visualize eye tracker data ? 19
1.3.6 Eye tracking metrics 20
1.3.7 Model Comprehension and Mental Models 26
1.4 Conclusion 27

CHAPTER 2 LITERATURE REVIEW 29
2.1 Studies on code comprehension 29
2.2 Studies on Model comprehension 30
2.3 Studies on Feature Models 36
2.4 Conclusion 40

CHAPTER 3 EMPIRICAL STUDY DESIGN 41
3.1 Objective 41
3.2 Research Questions 42
3.3 Task and Setting 43
3.3.1 Material 45
3.3.2 Stimuli Selection 46
3.3.3 Experimental Design 48
3.3.4 Participants 50
3.3.5 Experiment Execution 50
3.3.6 Eye-Tracker Specification and Setup 51
3.4 Data Extraction 53
3.5 Data processing and analysis plan 56
3.5.1 Interviews – Qualitative analysis 59
3.6 Summary 59

CHAPTER 4 RESULTS AND DISCUSSION 61
4.1 Descriptive Statistics 61
4.2 RQ1 results 63
4.3 RQ2 results 65
4.4 Interview Results 69
4.5 Threats to Validity 72

CONCLUSION AND RECOMMENDATIONS 75

BIBLIOGRAPHY 77

LIST OF TABLES

Page

Table 1.1 Tobii Pro Fusion requirements 17

Table 1.2 Tobii Pro Fusion specifications 18

Table 1.3 Tobii Pro Fusion setup 19

Table 1.4 Eye-tracking-metrics 23

Table 2.1 Studies using eye-tracking technology for model comprehension-Goal 37

Table 2.2 Studies using eye-tracking technology for model comprehension-variables 38

Table 2.3 Studies using eye-tracking technology for model comprehension-design 39

Table 2.4 Metrics use for evaluating structural complexity of feature models 40

Table 3.1 Feature Model Selection Summary 45

Table 3.2 24 assignments 47

LIST OF FIGURES

Page

Figure 1.1 Effort/cost of crafting products individually versus product-line development Taken from Apel, Batory, Kästner & Saake (2013) 5

Figure 1.2 Feature Models – (a) Mandatory feature, (b) Optional Feature, (c) Alternative group, (d) Or group, (e) basic Cross-Tree Constraints 7

Figure 1.3 Running Example – Laptop Product Line(Feature Model with our Running Example) 9

Figure 1.4 (a) head-free, (b) head-boxed, (c) head-restricted Taken from Valtakari *et al.* (2021) 16

Figure 1.5 Tobii-Pro-Fusion/120 HZ 16

Figure 1.6 Tobii-Pro-Lab/Platform 17

Figure 1.7 heatmap 21

Figure 1.8 GazePlot 21

Figure 1.9 Radial Transition Graph Taken from (Peterson, Saddler, Blascheck & Sharif (2019)) 22

Figure 1.10 Color Coded Taken from Busjahn, Schulte & Busjahn (2011) 22

Figure 3.1 Stimulus Example – Java graphical interface with superimposed Areas of Interest (AOIs): ① Answer — answer panel, ② Buttons — buttons panel, ③ CTC – cross-tree constraints area, ④ FM – feature model, ⑤ Legend – legend label, ⑥ Question – question panel, ⑦ Window – background window. 46

Figure 3.2 The process of experiment execution – (1) Educational video, (2) Calibration process, (3) Performing Tasks, (4) Interview 51

Figure 3.3 Experimental Room Layout 52

Figure 3.4 9 dots Calibration Process 52

Figure 3.5 Task window 53

Figure 3.6 Time of Interest (ToI) specified with Tobii Pro Lab 54

Figure 3.7 Naming process and assigning frame to ToI 54

Figure 3.8 Creating heatmap for each task 55

Figure 3.9 Assigning AoIs for each task 55

Figure 3.10 Diagram shows the steps of experiment from SPLOT repository until data extraction – Green: Public sections, Red: Private sections, Purple: research-based sections, Uncolored: Not automated sections, Blue: Eye-tracker software 58

Figure 3.11 Diagram shows the steps of data collection for each participant 58

Figure 4.1 Correct and Incorrect Responses per NoF and NoC ranges 62

Figure 4.2 Response Time in seconds for Correct Answers per NoF and NoC ranges 62

Figure 4.3 Average Proportions of Fixation Time and Count 63

LIST OF ABBREVIATIONS

ETS	École de Technologie Supérieure
ASC	Agence Spatiale Canadienne
NSERC	Natural Sciences and Engineering Research Council of Canada
SPLs	Software Product Lines
SPL	Software Product Line
SPLC	Software Product Conference
FM	Feature Model
NoF	Number of feature
NoC	Number of cross-tree constraints
CTC	Cross-tree constraints
SPLE	Software Product Line Engineering

INTRODUCTION

Feature models are the de facto standard to represent the features and their combinations whose implementations are used for constructing the set of similar software systems that constitute a *Software Product Line (SPL)*. Feature models play a crucial role throughout the development cycle of SPL Pohl, Bockle & van der Linden (2005); Apel *et al.* (2013); Capilla, Bosch & Kang (2013). Despite their importance, the study of comprehension of these models is an area that remains largely unexplored. One of the most common tasks performed with feature models is checking that a given configuration, i.e. a combination of features, satisfies or not all the constraints expressed by a feature model. In other words, checking whether a configuration is valid or not. Despite the enormous progress in automating the support for this and other similar analysis tasks Thüm, Apel, Kästner, Schaefer & Saake (2014); Galindo, Benavides, Trinidad, Gutiérrez-Fernández & Ruiz-Cortés (2019), all of them are ultimately driven by SPL developers who must look at a feature model and comprehend its meaning.

The goal of our empirical study is identifying some challenges involved in comprehending feature models, more concretely it focuses on the impact that the number of features and the number of cross-tree constraints (see definitions in Section 1.1) have on the comprehension of feature models for answering questions on the validity of configurations. More specifically, our study: *i)* explores the relation of these two feature model metrics with the accuracy of responses and the response time, and *ii)* uses eye fixations as an estimation of the cognitive effort taken by the different stimuli components of the tasks.

For our experimental design, we calculated the number of features and number of cross-tree constraints for 882 feature models taken from the largest repository available. We divided the whole dataset of feature models into four ranges for number of features and 3 ranges for cross-tree constraints, based on the distribution of these metrics along their quartile values. We obtained a random sample of 2 feature models for each of the twelve factor combinations, yielding a total of 24 feature models, for which we anonymized their feature names and devised a validity checking question of similar complexity. We recruited 17 participants, each of them followed a different random order of tasks.

Among other findings, our results show that the accuracy of the answers does not relate individually to the number of features or to the number of cross-tree constrains of a feature model. Instead, both of these factors show an interaction in accuracy. More strikingly, we found that looking more frequently or longer at the feature model increases the likelihood of providing an incorrect answer, irrespective of both the number of features and the number of cross-tree constraints. We argue that our work opens several avenues of research on the comprehension of feature models for configuration and other related tasks.

The thesis is based on the article *"Towards a Cognitive Model of Feature Model Comprehension: An Exploratory Study using Eye-Tracking"*, DOI https://doi.org/10.1145/3546932.3546995, accepted for publication at the 26th ACM International Systems and Software Product Line Conference (SPLC 2022). The paper is co-authored by the author of the thesis, Kambiz Nezami Balouchi, and two members of the thesis jury, Prof. Julien Mercier and Prof. Roberto Erick Lopez Herrejon.

CHAPTER 1

BACKGROUND

In this section, we present the basic concepts and terminology of Software Product Lines and Feature Models necessary to follow the results presented in the thesis. We introduce a running example that we use to illustrate and explain our study throughout the thesis, and we define the core terms related to eye-trackers studies in software engineering.

1.1 Software Product Lines

Software Product Lines (SPLs) are families of related systems whose members are distinguished by the set of features they provide Batory, Sarvela & Rauschmayer (2004); Pohl *et al.* (2005). The idea behind developing Software Product Lines is to use existing reusable software components instead of creating them from scratch. Software systems should be designed based on customer requirements, and customers have an option to select desired features from available repositories Apel *et al.* (2013). There exist an extensive body of research and application of SPLs principles that attests the benefits that SPLs provide (e.g., Pohl *et al.* (2005); Heradio, Perez-Morago, Fernández-Amorós, Cabrerizo & Herrera-Viedma (2016); Galster, Weyns, Tofan, Michalik & Avgeriou (2014); Chen & Babar (2011)).

A tangible example of SPL is operating systems, which exists in almost in all computing systems and act as mediator between hardware and software. The Linux Kernel is an example of an operating system that serves on different platforms such as embedded devices, large servers, mobile devices, desktop systems and many others. One can easily estimate that a single operating system cannot support all these different platforms. Instead, existing operating systems like Linux Kernel are software product lines that can be configured to serve different purposes (Apel *et al.* (2013)).

1.1.1 The goal of Software Product Line

The purpose of Software Product Line is to enhance the production of software systems. Instead of developing software systems from scratch, having a software product line from which many products can be extracted using existing software parts helps to adapt to market changes. Indeed, by creating SPLs the goal is to fulfill some major concerns which are hard to achieve by developing software from scratch (Apel *et al.* (2013)):

- *Tailor-made*: SPLs promise, apart from mass production, to offer customizations based on customer or market requirements.
- *Reduced costs*: Initiating a unique system that can support the derivation of various applications while providing individual customer requirements, will reduce the total cost of producing every single product compared to developing software systems from scratch. Although, the initial cost of establishing SPLs is unquestionably greater than developing a single software product from scratch, the SPLs approach compensate this cost in the long term as illustrated in Figure 1.1.
- *Improved quality*: SPLs improve the quality of mass production through standardized software components, frequently. In SPLs, the software assets that are used in many products are more stable and reliable because they have been tested many times. However, in a one-off software system, the testing process is tailored for this particular software. In other words, during the product development process in SPLs, all common software parts for every single configuration are tested. Therefore, checking these parts each time for each configuration improves the quality of the final product.
- *Time to market*: Because in SPL products are assembled using available and ready parts, they can be delivered faster to the market. In turn, one-off software products requires a considerable amount of time and as consequence costs. Also, in the case that customer requires a new option which is not available in the system, developing this new option and adding it to existing and well-designed product is much faster comparing to building a software system from scratch.

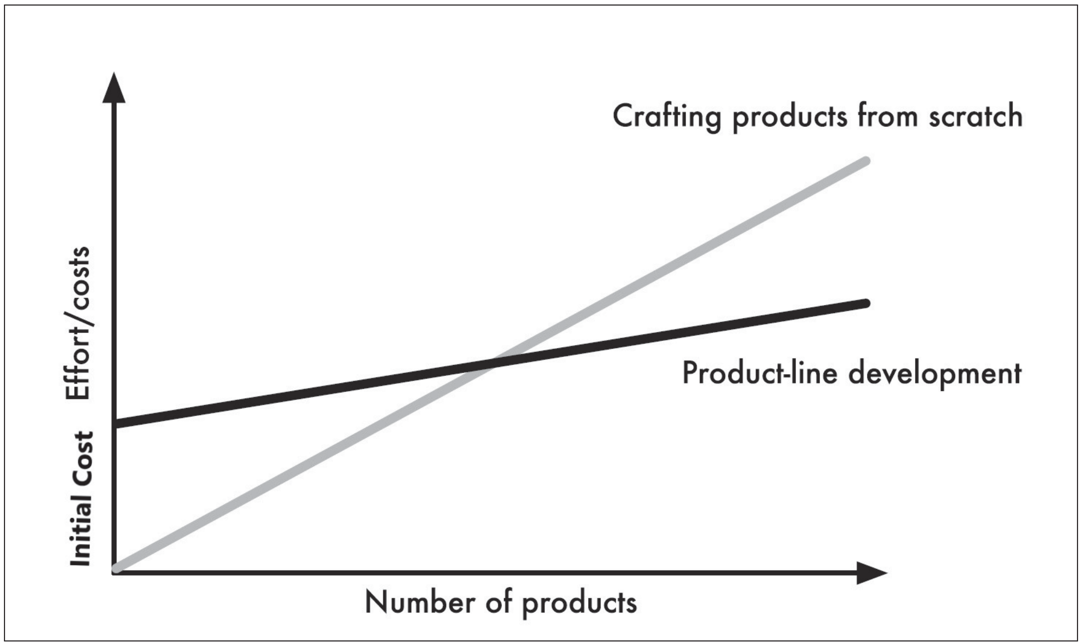


Figure 1.1 Effort/cost of crafting products individually versus product-line development
Taken from Apel *et al.* (2013)

Designing SPLs also has its own difficulties, like preparing reusable components at the beginning need significant investment. Additionally, having a large set of configurations that offer users a variety of options will require a lot of effort to maintain and manage.

1.1.2 Variability modelling approaches

In designing a Software Product Line, one of the key steps is *variability modelling* (VM). In software artifact analysis, the role of variability modelling is to identify and display the variability of a software artifact, establish and manage the dependencies among those variabilities, and allow them to be exploited for the building and evolution of a family of software systems. There exist different approaches to applying VM at different stages of designing SPLs based on the requirements and goals, such as *Feature model* , *UML and its extensibility*, *Express variability as part of a technique that models the architecture of the system*. The majority of studies applied feature models and extended UML to manage the variability modelling summarized by

Chen & Ali Babar (2011). All approaches follow a common goal to show variability in terms of common and optional features. Apel *et al.* (2013).

1.1.3 Introducing Feature Models

Software product lines apply a Feature-Oriented approach, in which features are at the core of assembling SPLs and represent various software assets in SPLs. To build the final product, different features (assets) are selected based on customers' needs. Then, selected features are assembled and validated in terms of compatibility before delivery.

Features carry the specific characteristics of its SPLs and show the commonalities and differences between all possible producible products of a SPL.

"*A feature is a characteristic or end-user-visible behaviour of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle* (Apel *et al.* (2013).)"

In a Software Product Line, each software product is characterized by a different combination of features. All the possible feature combinations (that correspond to all the software products) are expressed by *variability models* for which there are different alternatives (Czarnecki, Grünbacher, Rabiser, Schmid & Wasowski (2012)); however, feature models have become the de facto standard (Kang, Cohen, Hess, Novak & Peterson (1990)). In this type of model, features are depicted as labelled boxes and their relationships as lines, collectively forming a tree-like structure. The typical graphical notation and a basic textual notation of feature models are shown in Figure 1.2.

A feature is *selected* when it is present in a software product. To select desired features, we must consider defined notations among them. These notations are as follows:

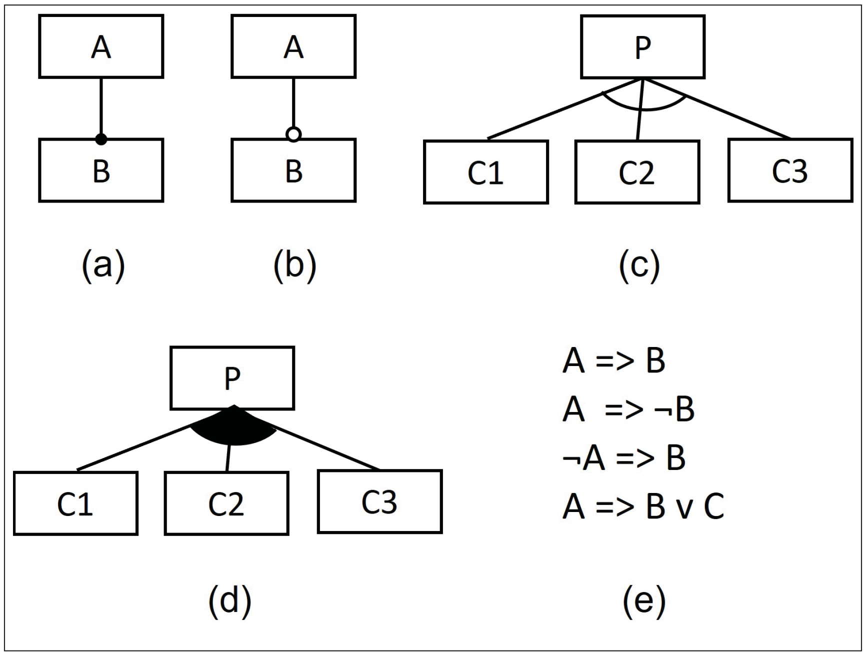


Figure 1.2 Feature Models – (a) Mandatory feature, (b) Optional Feature, (c) Alternative group, (d) Or group, (e) basic Cross-Tree Constraints

- *Mandatory* which is selected whenever its parent feature is also selected. For instance, in Figure 1.2(a), if feature `A` is selected, then feature `B` must be selected. Mandatory features are denoted with a filled circle on their end of the relationship line with their parent feature.
- *Optional* which may or may not be part of a product whenever its parent feature is selected. For example, in Figure 1.2(b), if feature `A` is selected, then feature `B` may or may not be selected. This means that there can be some software products with feature **B** and some others without it. Optional features are denoted with an empty circle on their end of the relationship line with their parent feature.

Features can be grouped into two ways:

- *Alternative group* whereby if the parent feature of the group is selected, *exactly one* feature from the group must be selected. For example, Figure 1.2(c), if feature `P` is selected, then exactly one of the group features `C1`, `C2` or `C3` must be selected. This means that the SPL with this group contains some software products with feature `C1` selected, some with feature

C2 selected, and some with feature C3 selected. However, no software product of this SPL can have more than one of these 3 features selected, as they are mutually exclusive. Alternative groups are denoted with an empty arc across the lines that relate the parent and the children features.

- *Or group* whereby if the parent feature of the group is selected, then *at least one* features from the group can be selected. For example, Figure 1.2(d), if feature P is selected, one or more features among C1, C2 or C3 must be selected. This means that the SPL with this group can have products with one, two, or three of these features selected. Or groups are denoted with a filled arc across the lines that relate the parent and the children features.

In addition to hierarchical parent-child relations explained above, features can also relate across different branches of the feature model with *Cross-Tree Constraints (CTCs)*(Benavides, Segura & Cortés (2010); Benavides (2019)). CTCs can be denoted graphically or using propositional logic formulas because it is a common underlying formal representation for the analysis of feature models (Benavides (2019)). We chose the propositional logic formulas because it is the format used by the FeatureIDE tool[1](Meinicke *et al.* (2017)), that we used to render the visual stimuli for our empirical study as explained in Section 3.3.2. All notations and cross-tree constraints of feature models can be represented as propositional logic formulas, which enable the final analysis and validation of feature models.

We employ only a few basic forms of constraints based on the random sample of feature models selected for our empirical study. More details on the selection of feature models is presented in Section 3.3. Our empirical study considers basic CTCs expressed with a logical implication of the form $\alpha => \beta$. We refer to the term α as the antecedent of the implication. Our study considers the following four forms:

- **A => B**. This form means that if feature A is selected, then feature B must also be selected.
- **A => ¬B**. This form means that if feature A is selected, then feature B must *not* (symbol ¬) be selected.
- **¬A => B**. This form means that if feature A is *not* selected, then feature B *must* be selected.

[1] https://featureide.github.io/

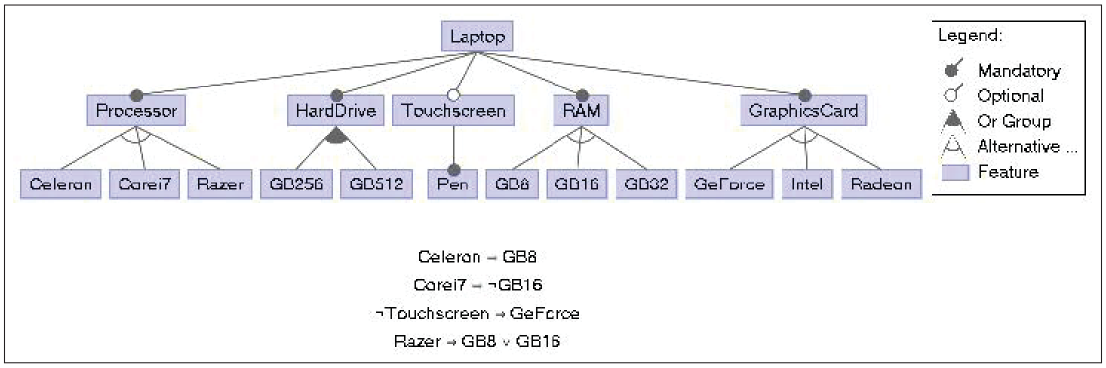


Figure 1.3 Running Example – Laptop Product Line(Feature Model with our Running Example)

- **`A => B`** ∨ **`C`**. This forms means that if feature `A` is selected, then (as denoted by the `or` operator) either feature `B`, feature `C`, or both must be selected.

Finally, because feature models are tree-like structures, they have a *root* feature that is always selected in all the products that conform a SPL. All above notations convey various parent-child relationships and constraints between features.

In the next subsection, we provide a running example to illustrate these concepts and our study design in Section 3.3.3.

1.1.4 Running Example Description

We use as a running example a fictitious product line of laptop configurations because it is a domain that is familiar to everybody. This example allows us to illustrate all the notations and concepts used in our empirical study without any loss of generality when applied to the realm of software.

Figure 1.3 shows the feature model of our running example. This figure was produced with the FeatureIDE tool. In this product line, a laptop computer is denoted with feature `Laptop` and has:

- A processor, denoted with mandatory feature `Processor`. In turn, the laptops can have only one type of processors, represented as an alternative group with the following features: `Celeron`, `Corei7`, and `Razer`.
- A hard drive, denoted with mandatory feature `HardDrive`. In turn, the laptops can have one or two hard drives, which is denoted by an or group with the following features: hard drive of 256GB represented by feature `GB256`[2], and hard drive with 512GB represented by feature `GB512`.
- Optionally has a touchscreen, denoted with optional feature `Touchscreen`. Hence, there are laptops in the product line that have touchscreens and some that have not. For those that have touchscreens, they come with a pen to write with, denoted with mandatory feature `Pen`. Hence, if a laptop has a touchscreen, then it must have a pen.
- RAM memory, denoted with mandatory feature `RAM`. The laptops can have one of three different memory sizes, which are denoted by an alternative group. The options are: 8GB represented by feature `GB8`, 16GB represented by feature `GB16`, and 32GB represented by feature `GB32`.
- A graphics card, denoted with feature `GraphicsCard`. The laptops can have only one type of graphics card, which is represented by an alternative group. The options are the following features that correspond to three types of graphics cards: `GeForce`, `Intel`, and `Radeon`.

In addition to the constraints that derive from the structure of the feature model, our running example also contains the following cross-tree constraints (CTCs):

1. `Celeron => GB8`. This CTC means that if a laptop has a `Celeron` processor then it must have 8GB of RAM, i.e. feature `GB8` must be selected.
2. `Corei7 => ¬GB16`. This CTC means that if a laptop has a `Corei7` processor, then it cannot have a RAM size of 16GB, i.e. feature `GB16` must *not* be selected.
3. `¬Touchscreen => GeForce`. This CTC means that if a laptop does not have a touchscreen (i.e. feature `Touchscreen` is *not* selected), then it must have a GeForce graphics card (i.e. feature `GeForce` *must* be selected).

[2] In FeatureIDE, the feature names cannot start with a number, therefore as our naming convention we start with the prefix GB all those features that refer to memory size in GB.

4. `Razer => GB8 ∨ GB16`. This CTC means that if a laptop has a `Razer` processor, then it can either have RAM of 8GB or (operator `∨`) 16GB, i.e feature `GB8`, or feature `GB16`, or both must be selected.

1.2 Full and Partial Configurations — Definitions and Examples

One of the principal tasks that require feature models is configuring a product, that is, defining which features are selected and which are not selected. This configuration task is the focus of our empirical study. Let us now provide working definitions and examples of the terminology we use in our study.

A *configuration* is defined as a set of features selected from a feature model. A configuration can be a ***full configuration*** if it considers all the constraints of an entire feature model, or can be a ***partial configuration*** if it only considers a subset of the constraints of a feature model. Recall that the constraints are those inferred from the hierarchical structure of the feature model (e.g. that only one feature from the alternative group of `Processor` can be selected) and those written as cross-tree constraints (e.g. if `Celeron` is selected then `GB8` must be selected).

Configurations can be *valid* or *invalid* if they satisfy or not their set of constraints. Thus, a ***valid full configuration*** meets *all* the constraints of an *entire* feature model and conversely, an ***invalid full configuration*** does *not* meet *all* the constraints of an *entire* feature model. Similarly, for partial configurations. A ***valid partial configuration*** meets *all* the constraints of a *subset* of the feature model and conversely an ***invalid partial configuration*** does *not* meet *all* the constraints of a *subset* of a feature model. The ***verification of the validity of a configuration***, be it partial or full, is defined as the process of checking that all the applicable constraints, hierarchical and cross-tree, are met or not. Let us now provide examples of the verification of the validity of four configurations (two full and two partial).

As a first example, please consider the following full configuration with the selected features `{Laptop, Processor, Celeron, HardDrive, GB256, Touchscreen, Pen, RAM, GB8, GraphicsCard, Intel}`. Let us first verify the constraints derived from the structure of the feature model. The root feature `Laptop` is selected, as well as all its mandatory child features: `Processor`, `HardDrive`, `RAM`, and `GraphicsCard`. Furthermore, optional feature `Touchscreen` and its mandatory child feature `Pen` are both correctly selected. Notice also that only one processor is selected (feature `Celeron`), one hard drive size is selected (feature `GB256`), one RAM size is selected (feature `GB8`), and one graphics card is selected (feature `Intel`). Now let us focus on the cross-tree constraints. The first CTC is met because feature `Celeron` is selected and `GB8` is also selected. The second CTC refers to `Corei7` in the antecedent of the implication, and because it is not selected (i.e. false) the constraint is met. Similarly, for the third CTC, because the `Touchscreen` feature is selected (i.e. true) the antecedent does not hold, and the constraint is met. Similarly, also for the fourth CTC whose antecedent is feature `Razer`, which is not selected and hence the constraint is met. Because this configuration meets all the constraints of the feature model, it is a valid full configuration.

As a second example, consider the following full configuration with the selected features `{Laptop, Processor, Corei7, HardDrive, GB512, Pen, RAM, GB16, GraphicsCard, Intel}`. Following a similar process as described for the hierarchical constraints of the first example, it is simple to verify that all these constraints are met except one. Namely, feature `Pen` is selected but its parent feature `Touchscreen` is not selected. This constraint violation alone is enough to render this configuration as an invalid full configuration. Nonetheless, let us analyze the cross-tree constraints. The first CTC is met because the antecedent of the implication refers to feature `Celeron` which is not selected. The second CTC is violated because feature `Corei7` is selected and `GB16` is also selected in the configuration, a situation that is prohibited by this CTC. The third CTC is also violated because given that feature `Touchscreen` is not selected, this constraint required that feature `GeForce` were selected, but it is not. The fourth CTC is met because the antecedent refers to feature `Razer` which is not selected. In summary, this full configuration is invalid because it violates three constraints, one from the hierarchy of the feature model and two cross-tree constraints.

As a third example, consider the following partial configuration with the selected features `Razer, HardDrive, GB256, GB512, Touchscreen, GB8, GeForce`. Because it is a partial configuration, for its validity we only need to verify the constraints where these features are involved. As before, we verify the hierarchical constraints. Feature `Razer` is selected and there are no other hierarchical constraints to check for it. Features `HardDrive`, `GB256`, and `GB512` belong to the same or group, hence the constraints of this type of group are met. `Touchscreen` is a feature that can be selected and there are no other constraints to verify, as neither the feature `Laptop` nor the feature `Pen` is in the set of features of this partial configuration. Similarly, for features `GB8` and `GeForce` there are no other hierarchical constraints to validate as the features they relate to are not selected in the partial configuration. Regarding the CTCs, the first CTC is relevant because it refers to feature `GB8`. However, because feature `Celeron` is not in the partial configuration (hence false), the CTC is met. The second CTC is not relevant, as it does not involve any of the features of the partial configuration. The third CTC is relevant because it refers to two features in the partial configuration. However, because feature `Touchscreen` is actually selected, the antecedent of the implication does not hold and hence the CTC is met. The fourth CTC is met because feature `Razer` and feature `GB8` are selected in the partial configuration. In summary, this is an example of a valid partial configuration, as it meets all constraints of its subset of the feature model.

As a final example, consider the following partial configuration with the selected features `{Corei7, Touchscreen, RAM, GB16, GB32, Intel}`. Again, because it is a partial configuration, we need to verify only the constraints where these features are involved. Feature `Corei7` is selected, and there are no other features that relate to it in the structure of the feature model. Similarly, feature `Touchscreen` is selected and none of the features it relates to are in the partial configuration. Thus, both features meet their subset of hierarchical constraints. Now left focus on the alternative group of feature `RAM` and the two members of the group, feature `GB16` and feature `GB32`, both of which are selected in the partial configuration. This is a clear violation of an alternative group, which requires that only one feature in the group can be selected. This violation alone is enough to render make the partial configuration invalid. Nonetheless, let us look at the remaining subset of constraints that need to be verified. Feature `Intel` is selected but no other feature it relates to are in the partial configuration, thus its constraints are met. Now, regarding the CTCs, the first one is not relevant because it does not reference any feature in the partial configuration. The second CTC is violated because `Corei7` is selected, but feature `GB16` is also selected. The third CTC is met because the antecedent of the implication does not hold, as feature `Touchscreen` is in the partial configuration. The fourth CTC is met because the antecedent of the implication is not in the partial configuration. In summary, this is an invalid partial configuration.

1.3 Eye-tracking Basics

There is an extensive and long-standing body of research in eye-tracking theory and practice (e.g. Duchowski (2017); Holmqvist & Andersson (2017)). Among this research, there is a vast number of works in the area of software engineering as summarized, for example, Obaidellah, Haek & Cheng (2018), and Sharafi, Soh & Guéhéneuc (2015); Sharafi *et al.* (2020). Eye-tracking is based on the visual attention of participants whose eye gaze data is recorded. The premise is that visual attention triggers the cognitive processes required for the comprehension and resolution of tasks, and in turn, such cognitive processes direct the visual attention to specific locations in the visual field of the participant.

1.3.1 What is eye tracking?

An eye-tracker is a sensor device that measures eye movement, eye position and pupil dilation to record the data that can be obtained from the subject of study when looking at stimuli. The series of eye movements are converted by eye-tracking algorithms into data that contains information such as gaze direction (x and y coordinates), fixation points, fixation duration, pupil diameter, mouse position, AOI hits, eye movement type and other metrics depending on the selected eye-tracking device and software used for analysis.

1.3.2 What are the different kinds of eye tracker?

Overall, eye trackers can be classified in three main setups (Valtakari *et al.* (2021)):

- *head-free* eye trackers like wearable eye-trackers, participants have freedom to move around. See Figure 1.4(a).
- *Head-boxed* eye trackers setup such as remote or stationary eye trackers required that eye tracker placed in a fixed position in front of participants with predefined distance. In experiments using stationary eye trackers, participants work with stimuli presented in a monitor. In this type of eye-tracker, participants can move their head slightly to the right and left, and this movement will not affect the accuracy of recorded data. See Figure 1.4(b).
- *Head-restricted* eye trackers have more limitations in terms of setting up and recording eye movements, as shown in Figure 1.4(c). Participants' chin fixed with a specific tool to prevent any head movement.

Depending on the experiment requirements, the choice of eye tracker for each research project can differ, and it is not simply a matter of preference. In addition to the above classification, eye trackers are also divided into two distinct categories: single and dual eye-tracking setups. Single eye-tracking setups refer to those devices by which only the gaze of one participant can be recorded however, in a dual eye-tracking set up the experimenter can record the gaze of two participants in parallel. In human interaction experiments, three main factors are in the focus of the researchers to select the proper eye tracker: i) The possibility of movement when participants performing task, ii) The quality of gathered data and iii) How easy to analyze the data. Valtakari *et al.* (2021).

To conduct this study, we used a *Tobii Pro Fusion* [3] eye-tracker and *Tobii Pro Lab* software [4] that comes with Tobii Pro Fusion to analyze and record eye movements. To employ Tobii Pro Fusion in the study, specific requirements should be met which are shown in Table 1.1.

[3] https://www.tobiipro.com/product-listing/fusion/

[4] https://www.tobiipro.com/product-listing/tobii-pro-lab/

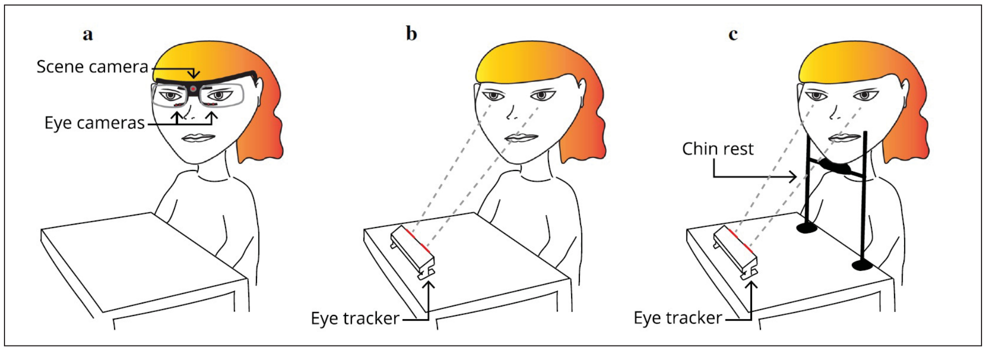


Figure 1.4 (a) head-free, (b) head-boxed, (c) head-restricted
Taken from Valtakari *et al.* (2021)

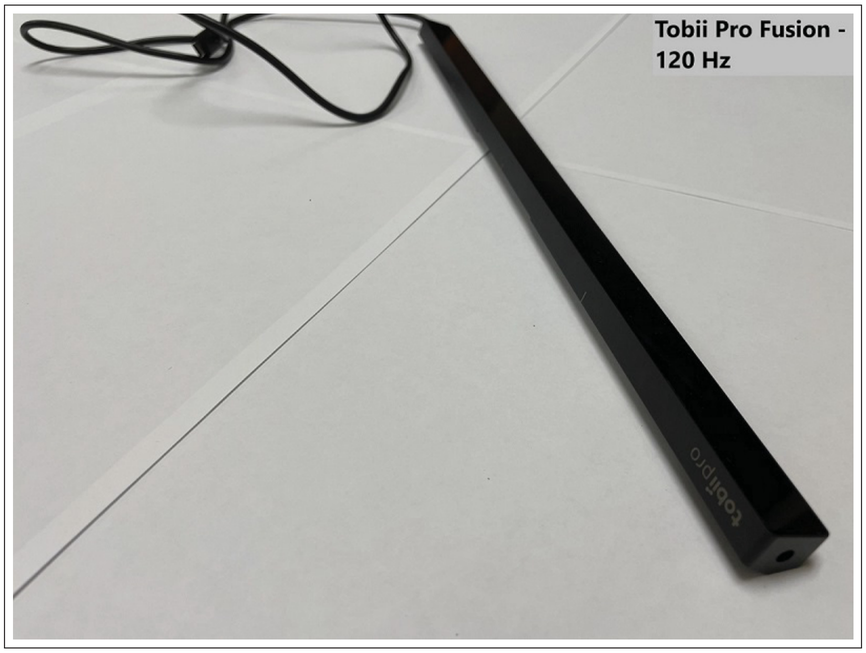
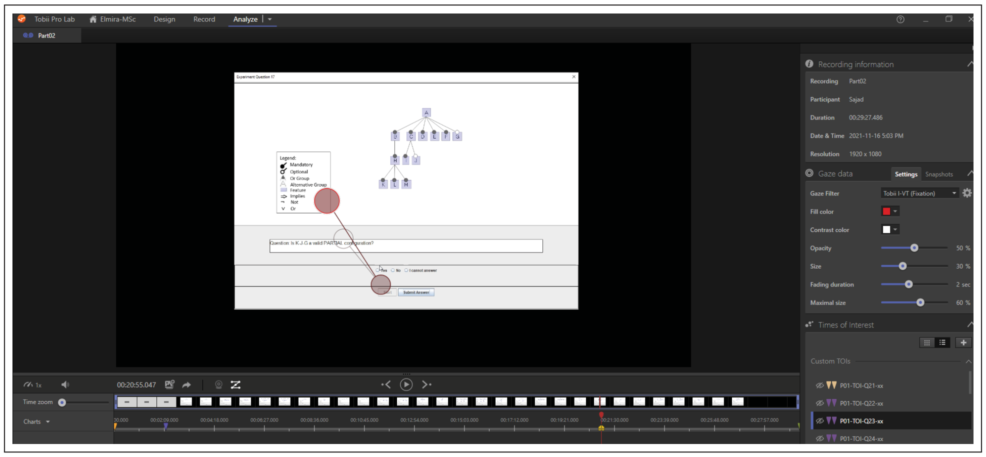


Figure 1.5 Tobii-Pro-Fusion/120 HZ

We show in Figure 1.5 and Figure 1.6 the eye-tracker version and the software environment (Tobii Pro Lab) we used in our experiment, respectively.



Recording information

Recording	Part02
Participant	Sajad
Duration	00:29:27.486
Date & Time	2021-11-16 5:03 PM
Resolution	1920 x 1080

Gaze data — Settings

Gaze Filter	Tobii I-VT (Fixation)
Fill color	🟥
Contrast color	⬜
Opacity	50 %
Size	30 %
Fading duration	2 sec
Maximal size	60 %

Figure 1.6 Tobii-Pro-Lab/Platform

Table 1.1 Tobii Pro Fusion requirements

Operating System:	Windows 10 macOS 10.14 Mojave and later
CPU:	1 GHz, 2 cores
RAM:	2 GB RAM memory Software for eye-tracking research may require higher RAM
Port:	USB Type-A or USB Type-C

1.3.3 How does it work?

Eye-tracking devices contain cameras, illuminators and algorithms. Eye-trackers use light sources to illuminate the eye. Near-infrared light is directed towards the center of the eyes (pupil) causing reflections in both the pupil and the cornea. These reflections can provide information about the movement and direction of the eyes, which are captured by a camera as an image. The captured images, are then processed with advanced image-processing algorithms to estimate the position of the eye in space and the point of gaze with high accuracy. It is worth mentioning that infrared light cannot be seen by human eyes, therefore it cannot distract participants during experiments.

1.3.4 What factors can influence data quality ?

In experiments using eye-tracking technology, having valuable results by which we can answer designed research questions, not only depends on the task, the experiment setup, and the process of selecting samples, but also highly depends on the quality of data we can extract from the eye tracker devices. The quality of eye tracker data can be influenced by many factors, including (Holmqvist, Nyström & Mulvey (2012)):

Table 1.2 Tobii Pro Fusion specifications

Hardware version:	120 HZ
Accuracy:	0.3° in optimal conditions
Data sample output:	Timestamp Gaze origin Gaze point Pupil diameter
Software options:	Pro Lab Pro SDK

- *Participants* are key elements in eye-tracking experiments. Each participant can have a different point of view for the same subject and a different methodology to follow instructions. Some participants may wear glasses, use contact lenses, have long eyelashes, mascaras or have droopy eyelids, which all may have an effect on the quality of images recorded by cameras.
- *Experimenters* have a different level of knowledge towards eye-tracking, and this may have an impact on the accuracy of recorded data.
- *Tasks* can affect the accuracy of recorded data, as some tasks may cause participants to blinks more than usual, therefore causing the data loss. However, blinks can be defined as an event in the experiment set up, and doing this can help differentiate blinks from other sources of data loss.
- *Environment and equipment* are among important factors that influence the precision of eye-tracker data. Depending on where the experiment is conducted (outdoor in sunlight or indoors in a controlled laboratory) the experimenter must control lights, noise and possible

vibration, especially for the stationary eye trackers which require that participants have a fixed position with a monitor and not move around while their eyes' movement are recorded.

- *Calibration process* demands carefully followed steps to get maximum data collection during eye movements.

According to the Tobii Pro User Manual, we followed recommended setup steps to calibrate the Tobii Pro Fusion eye-tracker used in our experiment. The key steps are outlined in Table 1.3. Additionally, we categorized Tobii Pro Fusion specifications in 1.2, which is also from the Tobii Pro User Manual.

Table 1.3 Tobii Pro Fusion setup

Freedom of head movement: (at 65cm distance)	Width × height: 40 cm × 25 cm (15.7" × 9.84") (At least one eye tracked)
Freedom of head movement : (at 80 cm distance)	Width x height: 45 cm x 30 cm (17.7” x 11.8”) (At least one eye tracked)
Operating distance:	50–80 cm (19.69”–31.49”) from the eye tracker
Tracker setup options: (mounted on screen)	Tracker mounted on tripod, allows for even larger screens and physical objects to be tracked.
Optimal screen size:	24" (16:9 aspect ratio)

1.3.5 How to visualize eye tracker data ?

The result of working with an eye tracker is a large table of data that can be visualized using different methods. Visualization enables researchers to understand the temporal and spatial aspects of eye movements extracted through the eye tracker as data points. In software engineering community most used visualization techniques are Gaze plots, Heat Maps, Colour Coded Attention Allocation Map and Radial Transition Graph. Here are brief descriptions of each visualization method and its sample figure.

- *Heatmap* A heatmap is a two-dimensional visual representation of data that presents the magnitude of a phenomenon with colors. Variations in color can be based on tone or intensity, giving the reader visual hints about where the phenomenon resides or varies over time[5]. In

[5] https://en.wikipedia.org/wiki/Heat_map

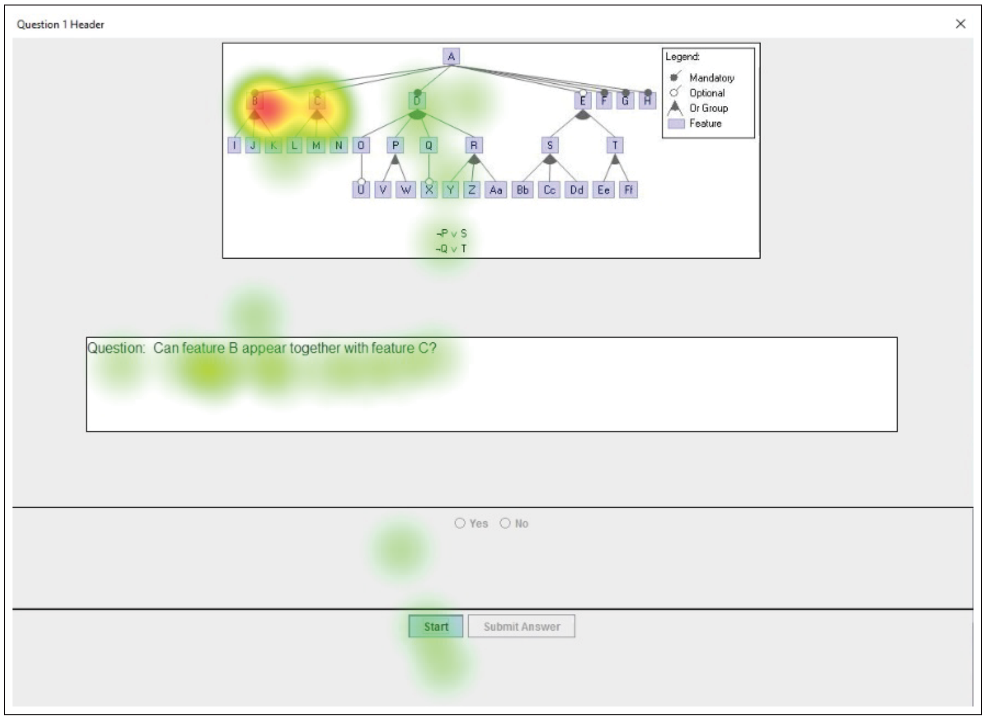
Figure /ref[fig:heatmap] moving from a darker area in a color (e.g., red) to a lighter area in a color (e.g., green) indicates participants' attention shifting from higher to lower. In fact, we can construct this spectrum based on the number and duration of fixations.

- *Gaze Plot* is an analytical technique offered by eye-tracking software shows gaze fixations on stimuli in the order in which they occur. Basically, the gaze plot shows where, when, and how long a person looked at an item. The diameter of the fixation circle indicates the amount of time spent looking. A larger circle indicates a longer look. Figure 1.8.
- *Radial Transition* is a circular map as shown in Figure 1.9. Circles in the maps present AOIs and the size of sectors change based on fixation duration in each AOI. The circles in the graph are colored according to the fixation number inside each AOI, and the arrows indicate the transition between them. In the figure, the number of transitions between AOIs is shown by the thickness of the arrow. Sharafi *et al.* (2020).
- *Color Coded* attention map assigns color for each word depending on fixation number or fixation duration. Words with the highest attention coded with red and those with the lowest attention coded with green. Assigning colors to the words follow the same pattern as Radial and heatmap techniques. Colors are select from green to red shades based on participants' attention, as shown in Figure 1.10.

To decide about the type of technique that best represents the data, we can refer to the experiment's tasks. For instance, Gaze plots can be used to identify and compare various participants' viewing strategies based on their scan paths. Heat maps are often used to show participants' attention distributions and examine visited AOIs either relevant or irrelevant. Sharafi *et al.* (2020).

1.3.6 Eye tracking metrics

In computer science and human-computer interaction studies including eye tracker, researchers adapt *performance measurement* as a testing method. The metrics used to measure the performance are *the number of fixation*, *fixation duration*, *Gaze switching behaviour* and *Scanpath*. All provides information about the cognitive state of participants during task performing Duchowski (2017). Here below, we list and define some common eye movement



Question: Can feature B appear together with feature C?

Figure 1.7 heatmap

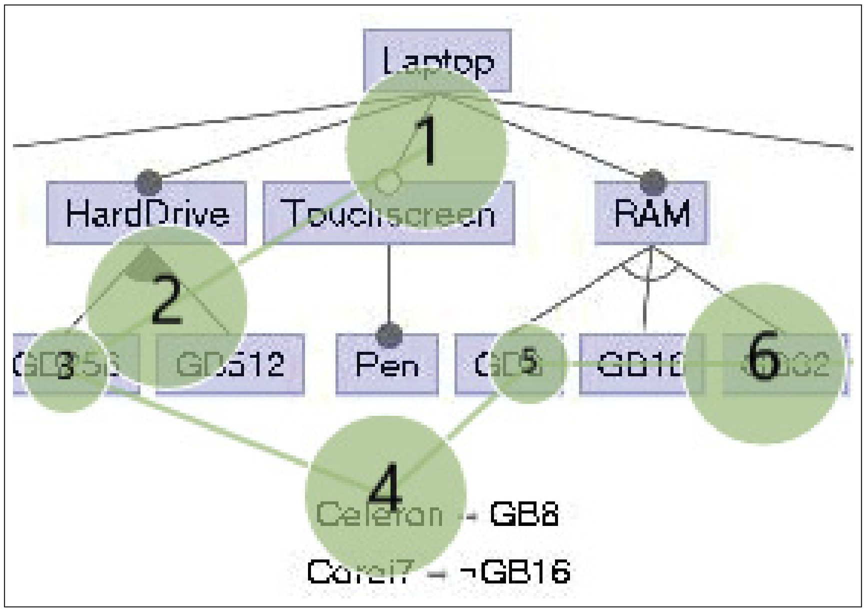
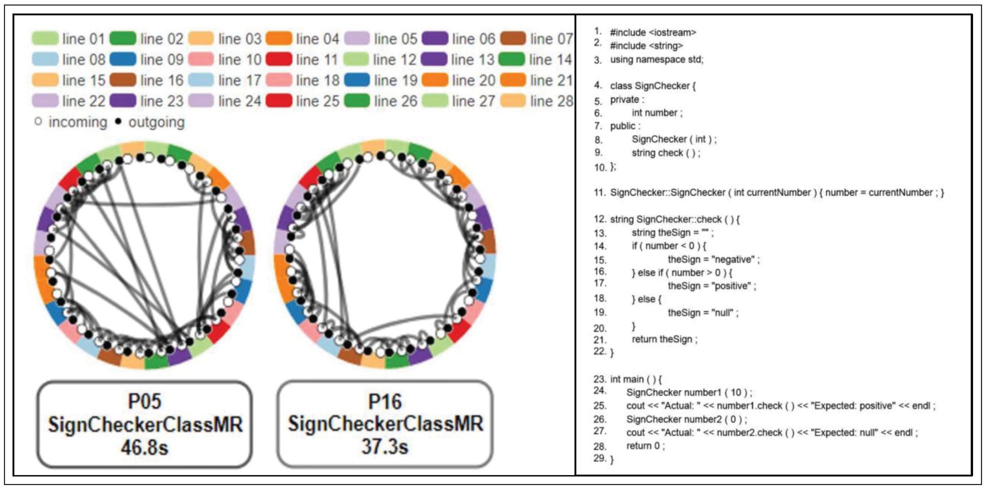
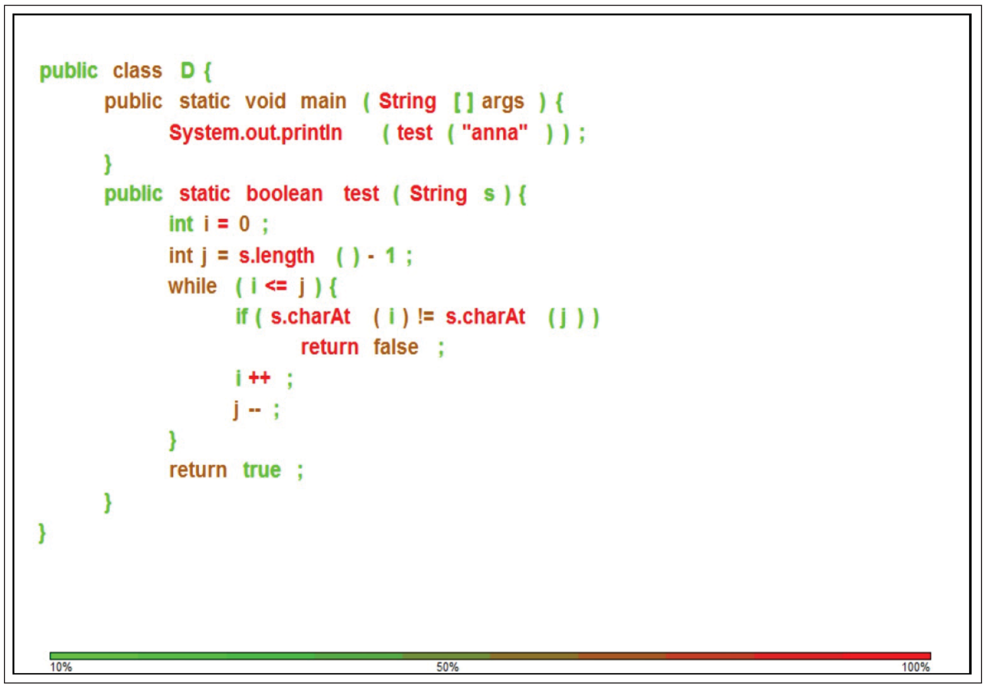


Figure 1.8 GazePlot



```
1.  #include <iostream>
2.  #include <string>
3.  using namespace std;

4.  class SignChecker {
5.  private :
6.          int number ;
7.  public :
8.          SignChecker ( int ) ;
9.          string check ( ) ;
10. };

11. SignChecker::SignChecker ( int currentNumber ) { number = currentNumber ; }

12. string SignChecker::check ( ) {
13.         string theSign = "" ;
14.         if ( number < 0 ) {
15.                 theSign = "negative" ;
16.         } else if ( number > 0 ) {
17.                 theSign = "positive" ;
18.         } else {
19.                 theSign = "null" ;
20.         }
21.         return theSign ;
22. }

23. int main ( ) {
24.       SignChecker number1 ( 10 ) ;
25.       cout << "Actual: " << number1.check ( ) << "Expected: positive" << endl ;
26.       SignChecker number2 ( 0 ) ;
27.       cout << "Actual: " << number2.check ( ) << "Expected: null" << endl ;
28.       return 0 ;
29. }
```

Figure 1.9 Radial Transition Graph
Taken from (Peterson *et al.* (2019))



```
public class D {
        public static void main ( String [ ] args ) {
                System.out.println ( test ( "anna" ) ) ;
        }
        public static boolean test ( String s ) {
                int i = 0 ;
                int j = s.length ( ) - 1 ;
                while ( i <= j ) {
                        if ( s.charAt ( i ) != s.charAt ( j ) )
                                return false ;
                        i ++ ;
                        j -- ;
                }
                return true ;
        }
}
```

Figure 1.10 Color Coded
Taken from Busjahn *et al.* (2011)

metrics use regularly in practice to show ocular behaviour Sharafi *et al.* (2020). We organized metrics based on classification provided by (Karn, Ellis & Juliano (1999) and Jacob & Karn (2003)) and summarized them in Table 1.4.

Table 1.4 Eye-tracking-metrics

First Order Data	X,Y position Pupil Diameter Eye blinks
Second Order Data	Fixations Saccades
Third Order Data	Fixation count Fixation duration or fixation time Time to the first fixation in an AOI(TTFF) Percentage of fixations or fixation rate Average Fixation Duration (AFD) All fixations within a selected time Ratio of On-target to All-target Fixations (ROAF) Saccade count Saccade duration or saccade time Regression rate
Fourth Order Data	Scanpath A convex-hull area Attention switching Fixation Spatial Density (SD) Linearity

First Order Data refer to row data gathered by eye-tracking devices when participants performing tasks on stimului such as:

- *X,Y position*: In an eye-tracking system, each gaze point is mapped to a location on the stimulus based on spatial coordinates.
- *Pupil diameter* refers to Pupil size, which can be varied in different participants. The size of pupil changes based on tasks' difficulty, and external factors such as: light, head movement, participant's emotional and cognitive state, the quality of camera provided by eye-tracking devices and distance to the eye-tracker. Data relating to pupil size can be used to evaluate the cognitive workload of participants, however, for meaningful results, experiments must be properly designed.

- *Eye blinks* refer to the number of blinks per minutes. Some eye-trackers do not provide eye blinks and in order to calculate metrics based on collected eye blinks data, researchers need advanced eye-tracker software and devices equipped with vision algorithms. Sharafi *et al.* (2020)

Second Order Data is obtained from first order data. The second order data refers to fixation and saccade metrics, which we define them below. The eye-trackers use event detection algorithms to differentiate fixations from saccades based on spatial and temporal criteria. Sharafi *et al.* (2020).

- *Fixations* are eye movements that stabilize the retina on an object of interest in a visual stimulus Duchowski (2017).
- *Saccade* is rapid eye movements ranging in duration from 10 to 100 ms. In other words, what happens between two fixations called *Saccade*.

Third Order Data is obtained by analyzing fixations and saccades in eye-tracking software. Some metrics are listed below:

- *Fixation count*: is a metric derived from the first order data and calculated by counting the number of fixations in a defined Area of Interests or whole stimuli.
- *Fixation duration or fixation time*: is the total amount of time calculated for all fixations in one AOI or stimuli.
- *Time to the first fixation in an AOI (TTFF)*: is the time taken from the beginning of the experiment until the participant's first gaze fixates on the AOI. This metric provides information about how different parts of stimuli are prioritized by participants.
- *Fixation rate* or *percentage of fixation*: is the ratio of the all number of fixation on one AOI to another. This metric shows which part of stimuli is most or least investigated by participants.
- *Average Fixation Duration (AFD)*: known as Mean Fixation Duration (MFD), represents the average fixation duration over all the fixations in an AOI, taking into account all the fixations counts in all the AOIs or stimuli. Duchowski (2017).
- *All fixations within a selected time* is obtained by counting the number of fixations in a particular AOI or stimulus within a specific period of time.

- *Ratio of On-target to All-target Fixations (ROAF)*: is computed by dividing the total umber of fixations in an AOI on the total number of fixations in the stimulus. A smaller ratio implies less efficiency due to the greater effort required to find the relevant elements needed for a task. Cepeda Porras & Guéhéneuc (2010).
- *Saccade count*: calculated by counting the total number of saccades in an AOI.
- *Saccade duration or saccade time*: the duration of all saccades in an AOI or stimulus.
- *Regression rate*: A measure of the proportion of saccades that are backward or regressive, such as leftward in left-to-right source-code reading. Sharafi *et al.* (2020).

Fourth Order Data : These data contain information about how participants visualize and how long it takes them to find an answer, according to Sharafi et al. paper. Scanpath is a metric placed in this category that provides information about participants' search strategies. Scanpath can be measured using different algorithmic tools. Including Scanpath, there are other fourth order data which we provide a brief definition for them as below:

- *Scanpath* is a series of gaze fixations when participants performing tasks. Indeed, Scanpath is a combination of Saccade and Fixation. With time, scanpaths naturally lengthen, making them difficult to analyze and compare. In order to examine them, the numbers and locations of fixations, along with their temporal order and duration, must be considered. Scanpaths can be studied using various algorithmic tools such as: Transition matrix, Scanpath recall, precision, F-measure, and ScanMatch. Sharafi *et al.* (2020)
- *A convex-hull area* contains all the participants' fixations within the smallest convex set. If the value is small, this implies fixations were close together, indicating participants spent less time looking for relevant sections in the stimulus. Sharafi *et al.* (2020)
- *Attention switching* calculate the overall number of switches happening between all defined AOIs in the research for a determined amount of time (e.g., One minute).
- *Fixation Spatial Density (SD)* indicates the spreading of fixations over the gridded stimulus. Assume that a stimulus is divided into equal squares, then SD is the number of visited cells, i.e., a cell that has been fixated at least once. Sharafi *et al.* (2020). Soh et al. used this metric to analyze participants' search effort. Soh *et al.* (2012).

- *Linearity* employ in the research to evaluate participants' search strategies. Busjahn et al. used this metric to measure how closely participants adapt a text's natural reading format in source code reading tasks given to the participants. Busjahn *et al.* (2011).

As defined by Sharafi et al., a *visual stimulus* is any object required to perform a task whose visual perception triggers the participant's cognitive processes to perform some actions related to the task Sharafi *et al.* (2020). Section 3.3.2 provides further details on the stimuli definition

Area Of Interest (AOI) is an area in the visual stimulus where the researcher is interested in gathering data because they are deemed relevant for performing the participants' tasks (Holmqvist & Andersson (2017)). AOIs are used to defined third-order data, according to Sharafi's et al. data classification Sharafi *et al.* (2020).

1.3.7 Model Comprehension and Mental Models

Recall that our study focuses on tasks that check the validity of configurations. From a cognitive perspective, these tasks are considered as *comprehension tasks* that build, manipulate, and compare mental models of different components that intervene in their resolution. The notion of *mental models* has different connotations as a result of the evolution of the research on the subject(Cárdenas-Figueroa & Navarro (2020)), ranging from a representation of reality on which cognition operates to the logic used to make decisions and learn. In the field of program comprehension, a mental model consists of two parts(Détienne (2001); Bidlake, Aubanel & Voyer (2020): *i)*) a *program model* that is constructed by programmers based on the structural knowledge of the code, and *ii)* a *domain model* that is an abstract representation of the code built using knowledge of the domain and the real-world situation of the program code.

The driving goal of our research is developing a *cognitive model* of the feature model comprehension that helps to explain the cognitive load, process, and effort of different SPL tasks that use feature models. We posit that such cognitive model contains the following three distinct mental models for the tasks of checking the validity of configurations[6]: *i)* mental model of the

[6] We acknowledge the contribution of Prof. Julien Mercier in the elaboration of these mental models.

feature model of the task, *ii)* mental model of the CTCs of the tasks, and *iii)* mental model of the configuration to validate in the task. Note that the first mental model is pictorial Seel (2017), i.e. the feature diagram and its visual notation, whereas the latter two are verbal Seel (2017), respectively the text of logic formulas of the CTCs and the text with the features of the configuration to validate.

From this perspective, a task that checks the validity of a configuration is performed via a sequence of steps that builds and operates on the three mental models identified. The challenge of this type of task is handling the *cognitive overload* caused by the manipulation of the information necessary to accomplish it successfully. *Cognitive load* arises from the fact that information is processed in working memory, which has a limited capacity (typically 4±1 items) and duration (Chen, Paas & Sweller (2021)). When these limits are surpassed, a *comprehension breakdown* occurs that can eventually lead to an incorrect answer or might demand an update or repair of a mental model (e.g., rechecking a CTC already validated). These difficulties and sources of mistakes can be analyzed empirically using eye-tracking data to explain the cognitive factors involved.

1.4 Conclusion

This chapter provides a brief introduction to Software Product Lines, feature models, and Eye-tracking technology. In each subject, we outlined preliminary concepts that would clarify the study's goal. It will make it easier for the reader to follow the steps of the experiment without having to consult external sources. The next chapter highlights a few studies relevant to our research and provides a short description of each one. We also compiled all the necessary data about each research design and presented it in the form of tables.

CHAPTER 2

LITERATURE REVIEW

In this section, we review the works closest to the focus of our study. Eye-trackers are used in several types of software engineering research to understand non-code (Models) and source code. The first step is to introduce some of these studies that looked at how developers interpreted source code. Following that, we provide a brief description of each of the papers conducted utilizing eye-tracking technology to experiment with model comprehension. In this section, we focus more on papers that involve models and provide more details about the rationale of the experiment. In the last section, we describe a couple of studies that explicitly addressed feature models, but did not apply eye-tracking technologies. The goal is to examine what kind of metrics are used to evaluate feature models.

2.1 Studies on code comprehension

There is an extensive and seminal line of work by Janet Siegmund and colleagues in the study of program comprehension in code with variability, studying aspects such as color, conditional compilation versus feature-oriented programming, or disciplined vs non-disciplined annotations (e.g, Feigenspan *et al.* (2013); Santos, do Carmo Machado, de Almeida, Siegmund & Apel (2019); Schulze, Liebig, Siegmund & Apel (2013)). However, to the best of our knowledge, for this line of work they do not employ eye-tracker measures. In contrast, Melo et al. performed an experiment to understand how developers debug programs with and without variability Melo, Narcizo, Hansen, Brabrand & Wasowski (2017). In code fragments with variability, they found increases in debugging time and in the number of gaze transitions between the use and definitions of fields and methods. Da Costa et al. studied C code with annotations and the impact of three refactorings in terms of program comprehension and visual effort da Costa *et al.* (2021). They found that two of such refactorings had an impact. In contrast with these works on variability, our study uses feature models, not source code, and our task is comprehension of configuration validity rather than code debugging or refactoring.

2.2 Studies on Model comprehension

In our study, we focus on feature model and examine how participants comprehend feature models through given tasks. Sharafi *et al.* (2015) conducted a systematic literature review to classify articles from 1990 to 2014 in which eye-tracking technology is used to study participants' cognitive processes. In total, they found nine papers work on model comprehension using eye-tracking device. Six out of ten papers studied UML class diagrams comprehensibility and other three papers studied ERD, TROPOS and BPNM diagrams (Cagiltay, Tokdemir, Kilic & Topalli (2013); Sharafi, Marchetto, Susi, Antoniol & Guéhéneuc (2013); Uddin, Gaur, Gutwin & Roy (2015)). None of these papers worked on feature model comprehensibility.

In 2006, for the first time, researchers used eye-tracking technology to examine participants' cognitive behaviour while analyzing UML class diagrams. In order to comprehend a program, software engineers mostly look at class diagrams and source code. Accordingly, Gueheneuc et al. look at eye-tracker data to analyze how software engineers use class diagrams to comprehend the program Guéhéneuc (2006). They introduced a new visualization tool to collect and present the data from an eye-tracker. The visualization tool is called *TAUPE data viewer*. It can calculate fixations, saccades, Areas of Interest, as well as highlight the most visited parts of a diagram using alpha composition [1]. Through analysis of eye-tracker data, they realized that participants did not use the relationship between classes to comprehend the program.

In 2007, Yusuf et al. Conduct an experiment to study how participants assess UML class diagrams. Their purpose was to discover which characteristics of UML class diagrams like colour, layouts, and stereotype are most effective for understanding software comprehension and design tasks by using the /textit [Tobii 1750] eye-tracker and a set of 27 comprehension questions. They analyzed the accuracy and response time of the answers by collecting gaze points of participants which provide information such as the number of fixations, saccades and scanpath. They showed that the characteristics of UML class diagrams were useful for exploring and navigating diagrams more effectively. Yusuf, Kagdi & Maletic (2007).

[1] https://en.wikipedia.org/wiki/Alpha_compositing

In 2009, Jeanmart et at. studied the impact of the visitor pattern on Program Comprehension and Maintenance with UML Class diagrams using an eye-tracker. They collected fixations and saccades to calculate different metrics like; Average Duration of Relevant Fixations (ADRF), Average Duration of Non-Relevant Fixations (ADNRF), and Normalized Rate of Relevant Fixations (NRRF) when developers perform comprehension tasks. They designed two series of tasks with and without presence of Visitor patterns to evaluate whether, using Visitor design pattern with different layouts, will improve participants' performance while performing modification and comprehension tasks. After analyzing data, they concluded that using Visitor pattern did not reduce the participants' efforts for comprehension tasks. On the other side, participants who were familiar with Visitor patterns and UML diagrams showed better result for modification tasks. Jeanmart, Gueheneuc, Sahraoui & Habra (2009).

Later on, in 2010, Cepeda and Gueheneuc. conducted empirical experiment on UML class diagrams to evaluate developers' performance for tasks related to design pattern comprehension. The goal was to explore the efficiency of various design pattern on understanding UML class diagrams. They, adapted three main visual representations and compare them with common visual representation: UML collaboration notation. Like the above eye-tracking experiments, they measured participants' performance, with the percentage of correct answer and the amount of effort participants spend for given tasks. Higher performance shows higher number of correct answers given by participants, while spending less effort. Using eye-tracker, they collect data by counting the number of fixations on AOI (Area of Interest) and AOG (Area of Glance) to evaluate participants' effort. Having fixation points enable them to calculate metrics such as: Average Fixation Duration (AFD), Ratio of "On-Traget:All-Target" Fixation Time (ROAFT) and Ratio of "On-Target:All-Target" Fixations (ROAF).The results showed that all studied representations can enhance the comprehension process depending on the purpose of tasks. For instance, the UML collaboration notation and pattern-enhanced class diagrams, are more efficient for locating classes involving a design pattern or stereotype-enhanced UML diagrams are more efficient for composition and role tasks (Cepeda Porras & Guéhéneuc (2010)).

Another eye-tracking study was conducted in 2010 by Sharif et al. to investigate how two different layouts (orthogonal and multi-cluster) for UML class diagrams, will facilitate the understanding of different design pattern roles. They replicated their previous study using alternative methods of data collection. The previous study used an online questionnaire to ask participants about the usefulness of layouts, and the current study used an eye-tracker to measure participants' cognitive behaviour performing related tasks with the same layouts. The goal of the current study is to see in what degree the result of the two experiments (one with questionnaire Vs, eye-tracking study) are close to each other. If participants'thoughts revealed the same information as their eyes, follow-on stimuli. To evaluate the effectiveness of these two layouts, researchers measured accuracy, time, and visual effort of participants for designed tasks. They collected two main eye data, fixations and saccades. By using these data points they calculated other metrics relevant to the goal of research such as: Fixation Count (FC), Fixation Rate (FR), and Average Fixation Duration (AFD) for Non-design and Design Pattern Classes. Both studies, showed that multi-cluster layout resulted in higher accuracy for role detection than orthogonal layout. In addition, participants spend less time and use less visual effort to find answers for all four design patterns presented in the multi-cluster layout. Sharif & Maletic (2010).

In 2012, Petrusel et al. conducted an eye-tracking experiment to determine how experts interpret business process models and which areas of the models are explored more to answer comprehension tasks. In addition, they found that by calculating the number of fixations and time spent in a certain area, they could predict whether participants provided the correct answer or not. They concluded that those participants, who answered comprehension tasks correctly, had longer fixation time on the relevant region elements than on other model elements and more elements are fixated in relevant region compared to other model elements. They showed that an answer is more likely to be correct if more time is spent fixating the relevant region elements (Petrusel & Mendling (2013)).

In 2012, Soh et al. studied how speed and accuracy of the subjects are related to their status and expertise in performing UML diagram maintenance. The purpose of the study was to provide project managers with an accurate perspective on how to recruit motivated engineers. They employed eye-tracking to collect fixation points of participants when performing comprehension tasks. They evaluated participants' comprehension by measuring accuracy, time spent, effort in searching, overall effort, and question comprehension effort. They recruited two groups of participants (practitioner vs. students) with two level of expertise (expert vs. novice). The analyses showed that practitioners are more accurate than students, while students spend less time than practitioner, a similar result was also found for experts vs. novices. Furthermore, they find out experienced students are more precise and spend less time compare to experienced practitioner Soh *et al.* (2012).

Cagiltay et al. in 2013 conducted an eye-tracking experiment to analyze software engineers' behaviour while designing, modifying, or comprehending Entity relationship diagram (ERD). Researchers proposed two measures for ERD defect detection process. They collected gaze data of participants while they were performing defect detection tasks. They also analyzed the relation between the performance of participants and their search pattern, within an ERD. The main motivation of the study is to have valid measures to evaluate how software engineers visualize and comprehend ER diagrams. As a result of this study, researchers, software companies, and educators would gain insights into how to improve the ERD reasoning process (Cagiltay *et al.* (2013)).

Sharafi et al. in 2013 used video-based remote eye-tracker to investigate the efficiency of graphical versus textual representations in modelling and presenting software requirements. They designed requirement comprehension tasks to measure participants' efficiency in terms of the accuracy (percentage of correct answers) and the amount of the time and effort they spend to find answers. For graphical representation, they used *TROPOS* model which is goal-based oriented modelling and visualize requirements through actor and goal diagrams. After collecting fixation and saccades, they counted the number of fixation for each defined AOI to measure the amount of effort spent by participants to answer the tasks. They also measured the spent time using an eye-tracking system. As a result, in terms of accuracy, they found no statistically significant differences between representation types. TROPOS graphic representations; however, required more time and effort from participants (Sharafi *et al.* (2013)).

In 2014, Smet et al. Developed *Taupe* tool to visualize and analyze eye-tracking data. The authors report that available software tools with eye-tracker devices are not open-source, and do not always offer extensions to integrate new sophisticated analyses seamlessly. Thus, they developed a software system called Taupe to help researchers visualize, analyze, and edit eye-tracker data. Two key features of Taupe are compatibility and extensibility, which means researchers can use the software with any eye-tracker data and customize the software for their own analyses. In this paper, researchers describe the design process behind Taupe and use three different experiments to validate and verify the provided functionalities mentioned above De Smet *et al.* (2014).

In 2015, Uddin et al. performed an eye-tracker-based study on code clone visualization (CCVs) comprehension. Researchers investigated how participants comprehend scatter plots, treemaps, and Hierarchical Dependency Graphs, all of which were generated by the clone visualization tool (VisCad). A code clone visualization is a graphical representation of the clone detection results obtained through the command line and graphical analysis tools. The purpose of the study was to determine which elements of visualizations such as colors, shapes, and object positions contribute most to understand and to identify patterns of usage for different groups. To answer comprehension questions, participants had to check three types of clone visualizations mentioned above. To analyze the collected data, researchers measured accuracy (the number of correct answer) and response time. The researchers concluded that observing search patterns depended on participants' expertise and that visualizations' elements were crucial for understanding CCVs Uddin *et al.* (2015).

In 2018, Obaidellah and Al Haek conducted another mapping study similar to that conducted by Sharafi et al. in 2015 that was focused on the use of eye-tracking in computer programming. In this SMS, researchers attempted to provide a comprehensive overview of eye-tracking studies in the software engineering domain in order to understand the cognitive processes of developers performing different tasks. In total, they identified 63 papers between 1990 and 2017 which all studied program comprehension and debugging using eye-tracking devices. Among 63 reported papers, only ten articles studied model comprehension (non-code) with eye-trackers. All the data from the collected papers, including experimental design, participant numbers, analyzed variables and metrics, tools, materials, and year of studies conducted, was compiled and presented in tables. Furthermore, researchers reported that majority of participants of these collected articles were students and faculty members and among other programming languages, Java language and UML class diagrams are the most commonly used materials. In addition, Tobii eye-trackers are the most widely used eye-trackers in various studies Obaidellah *et al.* (2018).

In 2020, Duarte et al. conducted a systematic literature review to explore the use of eye-tracking as a method of understanding process models. Researchers selected the 10 most cited papers in which the comprehensibility of process models was evaluated with eye-tracking devices. The major difference between Obaidellah et al. SMS in 2018 and Duarte et al. SLR is that in SMS researchers excluded papers searching only for comprehension factors in business process model, instead authors in SLR only focused on comprehension factors of a business process model. As part of this study, researchers summarized information about factors including metrics used in eye-tracking to measure business process models' comprehension, the process model notations usually evaluated in studies, and the contributions made by these studies to the application of eye-tracker technology to evaluate business process models' comprehension (Duarte, da Silveira, de Albuquerque Brito & Lopes (2020)).

A comprehensive guideline was provided by Sharafi et al. in 2020 for conducting eye-tracking studies in software engineering. In this paper, they provided a brief background about eye-tracking technology, discussed the reason and the best time to conduct eye-tracking experiment, introduced metrics associated with gaze points data, summarized studies used eye-trackers in software engineering, presented practical advice on how to design and prepare eye-tracking studies, discussed and classified statistical methods suitable for analyzing eye-tracker data, and presented a set of visualizations techniques. This article provides a foundation for those in the software engineering community who are interested in investigating the cognitive behavior of participants during their experiments using eye-tracking technology. Sharafi *et al.* (2020).

In 2021, Goncales et al. conducted a Systematic Mapping Study (SMS) to identify and categorize cognitive load measures in software engineering and highlight challenges for further research. The authors, investigated components and techniques used to measure cognitive load. They collected the information regarding the types of sensors used to measure cognitive load, the metrics calculated to process cognitive behavior, the purpose of studies, the designed comprehension tasks, artifacts, the number of participants, used research methods, and the name of venue that study has been published (Gonçales, Farias & da Silva (2021)).

2.3 Studies on Feature Models

Feature models are a way of representing commonalities and variabilities of different products in SPLs. There exist several studies in which the quality of feature models were evaluated using various quality metrics. Bezerra et al. did a series of study between 2015 and 2017 mainly investigated the effect of different metrics on evaluating the quality of feature models. Researchers in 2015 proposed that quality assessment of feature models at the beginning of SPL development can prevent errors later on in the process. To support the evaluation of feature models, they provided a catalog of metrics called COfFEE (Catalog of Measures for Feature Model Quality Evaluation). Later in 2016, various maintainability metrics were examined and their potential for use in assessing the quality of feature models. The authors concluded that by adding features over the course of SPL evolution, the structural complexity of feature models will increase and thus make them less modifiable and analyzable. In 2017, they studied the impact of different metrics taken from COfFEE on evaluating the quality of feature models. The authors adapted Spearman's rank correlation coefficient to check if metrics are correlated to each other. From the 32 proposed quality metrics in COfFEE catalog, only 15 are necessary to evaluate feature model quality. The following metrics have strong correlations with those 15 metrics. In other words, two different metrics which have a strong correlation with each other show the same results when evaluating the quality of feature models, and only one should be considered

Table 2.1 Studies using eye-tracking technology for model comprehension-Goal

papers	Goal of the research	Artifact	Eye-tracker
Guéhéneuc (2006)	How software engineers use class diagrams during the program comprehension activities	UML Class diagram	EyeLink II
Yusuf *et al.* (2007)	How human subjects use different types of information in UML class diagrams in performing their tasks	UML Class diagram	Tobii 1750
Jeanmart *et al.* (2009)	Determine whether the Visitor pattern is useful for comprehension and maintenance	UML Class diagram	EyeLink II
Cepeda Porras & Guéhéneuc (2010)	Effect of using representations Dong, Gamma, and Schauer, compared to representation UML, on the performance of developers while performing design pattern comprehension tasks	UML class diagram	EyeLink II
Sharif & Maletic (2010)	The effect of two layout schemes for class diagrams	UML class diagram	Tobii 1750
Petrusel & Mendling (2013)	To prove that the Relevant Region is indeed correlated to the answer given to the comprehension question	N/A	N/A
Soh *et al.* (2012)	Understanding the relation between the speed and accuracy of the subjects and their status and expertise in performing maintenance tasks on UML class diagrams and possible effect of the precision of the question on class diagram comprehension	UML Class diagram	EyeLink II
Cagiltay *et al.* (2013)	To develop measures to better understand performance of software engineers during their understanding process of ERD	ER (Entity Relationship) diagram	Tobii
Sharafi *et al.* (2013)	investigating the relations between the type of requirement representations (graphical vs. textual) and subjects' visual effort, required time, as well as accuracy in understanding requirements	TROPOS (is a goal-based oriented modeling approach)	FaceLAB
De Smet *et al.* (2014)	Impact of visitor design pattern on comprehension and modification Impact of Composite and Observer design pattern comprehension and modification Impact of MVC architecture style on maintenance tasks	UML class diagram	Eye-link II, Eye-link II, and FaceLAB
Uddin *et al.* (2015)	To find out what elements of the visualizations(e.g., colors, shapes, object positions) are most important for comprehension, and to identify common usage patterns for different groups	Graphics (Code clone visualizations)	Tobii T60 XL

for evaluation (Bezerra, Andrade & Monteiro (2015), Bezerra, Monteiro, Andrade & Rocha (2016) and Bezerra, Andrade & Monteiro (2017)).

Table 2.2 Studies using eye-tracking technology for model comprehension-variables

papers	**Independent variable**	**Dependent variable**
Guéhéneuc (2006)	N/A	N/A
Yusuf *et al.* (2007)	Layout (Orthogonal, Three-cluster, and Multiple-cluster)	Accuracy of answers, Response time of answers, and effort
Jeanmart *et al.* (2009)	Different Layout of visitor pattern	developers' effort (amount of attention)
Cepeda Porras & Guéhéneuc (2010)	Representations of variables and tasks	Fixation time
Sharif & Maletic (2010)	Reading behaviour	Accuracy, time, relevance, and visual effort
Petrusel & Mendling (2013)	N/A	N/A
Soh *et al.* (2012)	Status (practitioner, student) and Expertise (expert, novice) and Question precision (precise, not precise)	the average accuracy, the time spent, the search effort, the overall effort spent, and the question comprehension effort
Cagiltay *et al.* (2013)	Search pattern	Performance and defect detection difficulty
Sharafi *et al.* (2013)	Representation type (Graphic vs. text)	Accuracy, Time, and Effort
De Smet *et al.* (2014)	Visitor design pattern - Composite pattern Observer pattern - Different variants of MVC design pattern	Average number of fixation, Average duration of fixation effort - Spatial density, Transitional matrix, Average fixation's duration ON target / All target - Subject's speed, Subject's accuracy
Uddin *et al.* (2015)	Performance, accuracy, and response time	Expertise, and type of representation (Scatter plot, Treemap, Hierarchical Dependency Graph (HDG))

Vyas et al. conducted an empirical experiment in 2016 that was similar to what Bagheri et al. conducted in 2011. Bagheri et al. proposed a series of structural metrics and implemented validation processes for three main sub characteristics of maintainability (analyzability, changeability, and understandability) of the feature model. While Vyas et al. validated existing quality metrics for three main sub characteristics of usability attributes (learnability, understandability, and communicativeness) of the feature model. The both employed different statistical correlation methods were implemented to assess inter-metric correlation, inter-quality correlation, and metric indicativeness (Vyas & Sharma (2016) and Bagheri & Gasevic (2011)).

Table 2.3 Studies using eye-tracking technology for model comprehension-design

papers	Participants	Research design	Tool	measurements
Guéhéneuc (2006)	12	Within-Subject	The Taupe Data Viewer (Visualization technique)	N/A
Yusuf *et al.* (2007)	12	Within-subject	N/A	Accuracy and response time
Jeanmart *et al.* (2009)	24	Between-subject	N/A	subject's attention in an area of the screen
Cepeda Porras & Guéhéneuc (2010)	24	between-subjects	N/A	performance (the percentage of correct answers) and developers' effort
Sharif & Maletic (2010)	15	within-subjects	ClearView	Accuracy, time, relevance, visual effort using fixation points
Petrusel & Mendling (2013)	26	N/A	N/A	scan-path precision (SPP), scan-path recall (SPR), scan-path F-measure (SPF), and scan-path F2-measure (SPF2)
Soh *et al.* (2012)	21	Within-Subject	Taupe	subjects' comprehension with fixation Related metrics
Cagiltay *et al.* (2013)	5	N/A	ClearView	the time taken by participant and the defect detection order by participants
Sharafi *et al.* (2013)	28	Within-subject	Taupe	percentage of correct answers, time and effort spend to perform the tasks by calculating: Average Fixation Duration (AFD) and the surface of the AOI convex hull
De Smet *et al.* (2014)	24 - 24 - 23	between-subjects	Taupe	Norm-Rate, ADRF, and NRRF SD ,TM, ADRF, and IN-Alli Average time and Correctness
Uddin *et al.* (2015)	20	Within-subject	VisCad	participants' performance, accuracy and response time taken by participants to answer 25 stimuli questions using the audio and video recordings

El-Sharkawy et al. conducted a systematic literature review of available metrics for evaluating variability models, code artifacts, and metrics include both kinds of artifacts. They categorized all the metrics and papers from 2007 to 2016 in which for the first time the metrics introduced. The authors provided an overview of all the metrics and their formulas, and some examples of where they incorporated each. Based on a review of 42 papers, they identified 57 metrics only in variability modelling. As a result of their SLR, they created a catalog that provides variability-aware metrics for researchers and practitioners in the SPLs domain. (El-Sharkawy, Yamagishi-Eichler & Schmid (2019)). To select metrics of our experiment, we also considered El-Sharkawy et al. work to check available metrics used to assess feature models. Considering that two adapted metrics in our study use to measure structural complexity of feature models, we listed the metrics using to calculate structural complexity of feature models in table 2.4. These metrics and their functionality were evaluated by various studies such as Vyas & Sharma (2016) and Bagheri & Gasevic (2011).

Table 2.4 Metrics use for evaluating structural complexity of feature models

metrics	**Full Name**	**Description**
NoF	Number of features	Counting the number of features in the model
N_{Top}	Number of top features	Number of root descendants
CC	Cyclomatic complexity	Counting the number of distinct cycles in a feature model
NoC	Number of Cross-tree constraints	counting all cross-tree constraints
CoC	Coefficient of connectivity—density	Number of Edges divided on Number of features
DT	Depth of tree	Number of features of the longest path from the root of the feature model
$C_{om}C$	Compound Complexity	Measuring the cognitive complexity of a feature mode

2.4 Conclusion

From 1990 to 2017, two mapping studies were published on the use of eye-tracking technology for software programming, but none reported experiments on feature model comprehension. Moreover, we found that the number of studies involving model comprehension was ten in the same period, and all them were listed in both mapping studies except one by the Uddin et al. study done in 2015, which was not included in the Sharafi et al. mapping study due to the year of publication. Table 2.1, 2.2, and 2.3 summarizes the metrics, tools, experimental design, and materials used in each of these ten studies. In our summary, we exclude the SLR and SMS studies. To develop the experimental design for our eye-tracking experiment, we refer to the Sharafi *et al.* (2020) framework and the Systematic Mapping Study conducted by Gonçales *et al.* (2021) in which they classified cognitive load measures.

CHAPTER 3

EMPIRICAL STUDY DESIGN

Scientists have researched for decades how to identify patterns used by software developers to understand pieces of code, models, and debug code. Conducting controlled experiments allows researchers to study the cognitive processes that occur when software developers are processing information. Designing a reliable controlled experiment, needs to fully understand its steps and implementation, considering this we followed Siegmund et al. experimental design in her thesis in which they worked on establishing a unique framework for measuring program comprehension using controlled experiments Siegmund (2012).

We employ controlled experiment to analyze how software developers comprehend feature models use for presenting product derivation process in SPL. In this section, we define the research objective, state our research questions and provide their rationale, describe the process to select the feature models of our study, and how we constructed the visual stimuli from them. Furthermore, we describe our experimental design, including the selection of participants and assignment of experimental conditions.

3.1 Objective

As mentioned before, there is an extensive research on metrics of feature models El-Sharkawy, Yamagishi-Eichler & Schmid (2019). Our empirical study focuses on metrics used for evaluating structural complexity of feature models, see Table 2.4. Among them, we selected two of the most basic metrics, namely, *Number of Features (NoF)* and *Number of Cross-Tree Constraints (NoC)* to select the experiment's feature models. In this study our concern is to examine how software developers comprehend various feature models, which are different in number of features and the number of cross-tree constraints. We assume that feature models which have too many features and cross-tree constraints take more time for participants to answer the comprehension tasks. We hope that by adapting eye-tracking technology, we will be able to visualize the pattern participants follow and obtain the time they spent to find an answer. Therefore, our research goal is described as follows:

Goal. To analyze the impact of the metrics *Number of Features (NoF)* and *Number of Cross-Tree Constraints (NoC)* in the comprehension of feature models for the task of verifying the validity of full and partial configurations.

Henceforth, for sake of brevity, whenever we refer to *verification tasks* we imply the verification of the validity of configurations.

3.2 Research Questions

To reach our research goal, we define the following research questions and explain briefly what is the rationale behind each question.

RQ1. Is there any effect of NoF and NoC on the performance of verification tasks?

We break down performance of the verification tasks in two aspects: their accuracy that relates to the correct or incorrect responses, and their response time, which is measured from the moment the task is displayed on the screen to the moment when the participant enters his/her response. The corresponding questions are then as follows:

- ***RQ1a. Is there any effect of NoF and NoC on the accuracy of verification tasks?***
- ***RQ1b. Is there an effect of NoF and NoC on the response time of verification tasks that were answered correctly?***

Rationale: We postulate that higher values of the metrics NoF and NoC may increase the complexity of the creation and manipulation of the mental models of the feature model and the CTCs, and could consequently lead to comprehension breakdowns manifested as incorrect answers or longer times to respond. Thus, in this context, RQ1a investigates accuracy of tasks as the two metrics vary. Note that to further explore the cognitive processes required for a task, it is fundamental to focus on the tasks that were performed correctly, that is, those whose answers were correct. Therefore, RQ1b analyzes the response time of correct answers as the two feature model metrics vary.

RQ2. Is the accuracy of verification tasks related to the allocation of visual attention among the different components of the visual stimuli?

Similar to RQ1, we break down the allocation of visual attention in two aspects, namely, the amount of visual attention based on the fixation time metric, and the shifts of visual attention based on the fixation count metric. The corresponding questions are as follows:

- ***RQ2a. Is the accuracy of verification tasks related to the proportion of the amount of visual attention (fixation time) among the different components of the visual stimuli?***
- ***RQ2b. Is the accuracy of verification tasks related to the proportion of shifts in visual attention (fixation counts) among the different components of the visual stimuli?***

Rationale: As detailed in Section 3.3.3, we identified seven components in the visual stimulus of each verification task. For each of these components, we defined an Area of Interest (AOI) to analyze the allocation of visual attention among them. The fixation time on an AOI is the most global indicator of the difficulty of constructing the mental model associated with the corresponding visual component. Hence, RQ2a investigates using AOI data how accuracy of tasks changes as the two feature model metrics NoF and NoC vary, and sheds light on the complexity of their corresponding mental models. Fixation count on an AOI is an indicator of the effort devoted to understanding the various components of the visual stimulus. Therefore, RQ2b studies how this effort changes as the two feature model metrics NoF and NoC vary and its impact on the accuracy of responses.

3.3 Task and Setting

For our empirical study, we obtained all the 882 feature models available from SPLOT repository[1], which has the largest dataset of publicly available feature models. With the provided SPLOT API, we implemented the feature metrics NoF and NoC and applied them to our dataset. Table 3.1(a)

[1] http://www.splot-research.org/. Accessed on July 2021.

shows the minimum, maximum, and mean values as well as the first, second and third quartile values of our metrics NoF and NoC. On closer inspection, we observed a long-tailed distribution of values of NoC. Based on existing experience using feature models, we decided to set the maximum value of NoC at 15 for our study therefore, we eliminated feature models with more than 15 number of cross-tree constraints because above this number would make tasks more complicated and takes more time for participants to find answers. Hence, this will prolong the experiment session and cause participants to get tired after spending a determined time. To obtain reasonable experiment time, we run two preliminary pilot study with two graduate students with different fields, and we use provided guideline for eye-tracking experiments by Sharafi *et al.* (2020).The defined threshold for NoC eliminated only 35 feature models out of the original 882 models in our dataset.

We then analyzed the distribution of the feature models along the quartile values for the NoF metric. The ranges considered were: (1) Min<=NoF<Q1, (2) Q1<=NoF<Q2, (3) Q2<=NoF<Q3, and (4) NoF>=Q3. For the NoC metric, we set three ranges: (1) feature models with no CTCs (i.e. NoC=0, between minimum and Q1), (2) feature models with 1 or 2 CTCs (i.e. Q1<=NoC<Q3), and (3) feature models with 3 to 15 CTC (i.e. Q3<=NoC<=15).

Table 3.1(b) shows the number of feature models for each range combination of NoF and NoC. The highest frequency with value 114 was for feature models with between 20 and 34 features (range 3 in NoF dimension) and without CTC (range 1 in NoC dimension). The lowest frequency with value 29 was for feature models with 10 to 13 features (range 1 in NoF dimension) and 3 to 15 CTC (range 3 in dimension NoC). As a final step, for our empirical study, we randomly selected two feature models for each combination of ranges along both dimensions[2]. The details of the 24 feature models selected and their NoF and NoC are in our replication package.

[2] The feature models and the source code used for the random selection are available in an open access repository, Scholars Portal Dataverse via the following link: https://doi.org/10.5683/SP3/HUOVO3

Table 3.1 Feature Model Selection Summary

(a) Descriptive statistics, N=882

Metric	Min	Q1	Q2	Mean	Q3	Max
NoF	10	14	20	27.95	35	366
NoC	0	0	1	3.531	3	246

(b) Feature models per range of NoF and NoC, N=847

		NoF			
		(1) 10..13	**(2) 14..19**	**(3) 20..34**	**(4) >=35**
NoC	**(1) 0**	106	93	114	78
	(2) 1..2	73	88	40	34
	(3) 3..15	29	27	77	88

Ranges for NoF:1,2,3,4 Ranges for NoC: 1,2,3

3.3.1 Material

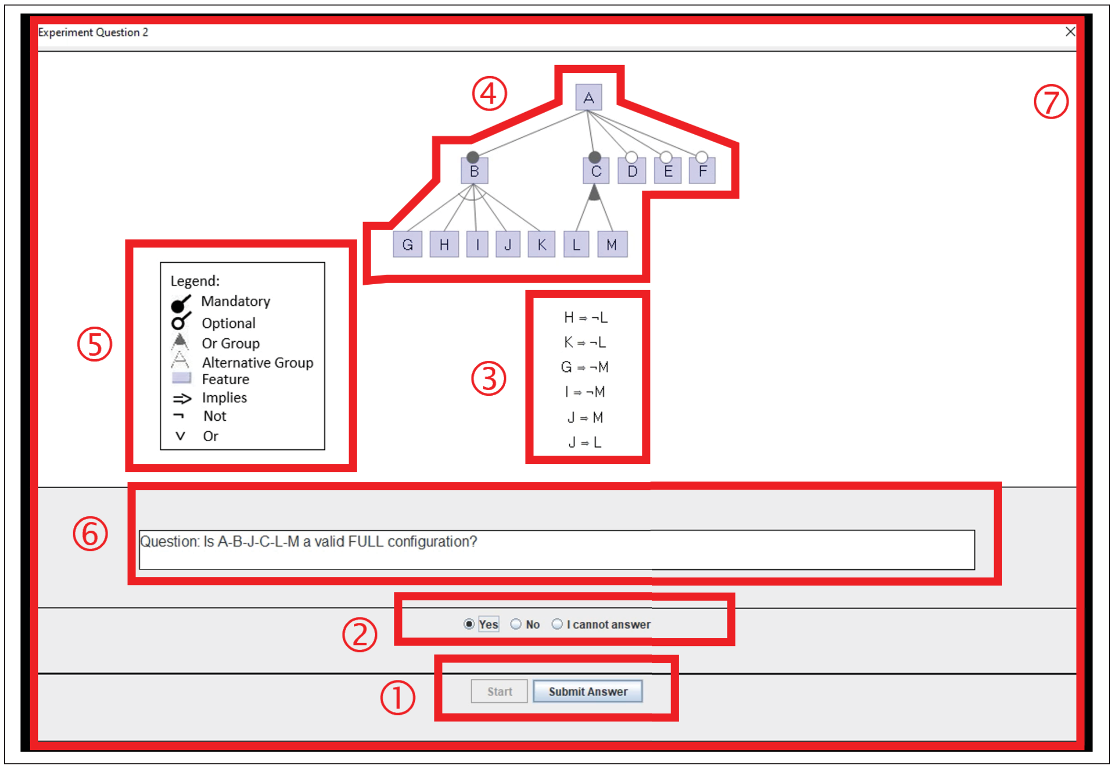
We explored different alternatives to present the stimuli of our study, among them iTrace[3] and PsychoPy[4]. However, it was not possible to link FeatureIDE, which is an Eclipse plugin, with them to allow us to obtain the information required for our study. Therefore, we wrote a bespoken Java program that presents a configurable sequence of questions, like that of Figure 3.1.

The program provides a simple user interface with the following elements: *1)* the feature model, *2)* a legend that summarizes the notations[5], *3)* area for cross-tree constraints of the form presented in Section 1.1, *4)* a question panel where the question is presented, *5)* an answer panel that shows the three possible answers `Yes`, `No`, and `I cannot answer` (given to participants to indicate that they were not able to respond), *6)* a response panel with the buttons to start answering the question and submitting the answer, and *7)* window-background window. The program records the responses and the response time. The feature model appears once the participant hits the `Start` button and the response time clock gets started.

[3] https://www.i-trace.org/

[4] https://www.psychopy.org/

[5] We manually extended the legend provided by default by FeatureIDE to include the notation of CTCs and placed it at the same position in all the questions.



Question: Is A-B-J-C-L-M a valid FULL configuration?

[✓ Yes] No I cannot answer

Figure 3.1 Stimulus Example – Java graphical interface with superimposed Areas of Interest (AOIs): ① Answer — answer panel, ② Buttons — buttons panel, ③ CTC – cross-tree constraints area, ④ FM – feature model, ⑤ Legend – legend label, ⑥ Question – question panel, ⑦ Window – background window.

The response time ends when the participant hits the `Submit Answer` button, which also clears the feature model for the next question.

3.3.2 Stimuli Selection

We define a task for each feature model in our sample. A task consist of answering a validity verification question that has one of the following two forms, where <*configuration*> stands for a list of features in a feature model:

- Is <*configuration*> a valid partial configuration?[6]
- Is <*configuration*> a valid full configuration?

Please do notice that in feature models, the names of the features are meaningful as they usually convey additional information about the relationships among features. This information could be used while executing tasks with the feature models, and it may have a positive or negative effect on the performance depending on the familiarity of the participants with the domain of the feature models. Hence, we anonymized the features' names to prevent any such effect by renaming the features with sequences of letters in alphabetical order and increasing length following a breadth-first traversal order of the feature model.

For the selection of the questions, we considered a length of six features and favored full configurations over partial configurations. In some cases, it was necessary to consider more or less features to provide meaningful questions. In summary, 17 questions use a partial configuration and 7 questions a full configuration. Regarding the number of features in the configurations, 14 questions have 6 features, four questions have 7 features, two questions have 3 features, two questions have 8 features, one question has 5 features, and one question has 10 features. See Table 3.2 .

Table 3.2 24 assignments

	Full configuration = 7	**Partial** configuration = 17
Valid:	**all** constraints are **met**	a **subset** of constraints are **met**
Invalid:	**Not all** constraints are **met**	**Not** a **subset** of constraints are **met**

We also aimed at balancing the distribution of the features that are part of the configuration question across the feature models. This means that we selected the features of the configuration questions in different areas along the horizontal axis (i.e., left, middle, right) and the vertical axis (i.e., top, middle, bottom). As an example, Figure 3.1 shows a full configuration question involving 13 features and 6 CTCs. Notice also that, for readability, we separated the features in the configuration with a dash (–) rather than with commas. In this figure, three of the features

[6] One question of this type has a different wording that requires the completion of the configuration.

are leaves on the tree-structure, two are children of the root feature, and half of the features appear to the left of the root and half to the right.

3.3.3 Experimental Design

We followed the guidelines proposed by Sharafi et al. for planning and executing our empirical study Sharafi *et al.* (2020). We chose a within-subjects design where each participant performed the 24 tasks, each task being a validity verification question as described above. In addition, we randomized the order of the tasks in such a way that each participant followed a different order (Lazar, Feng & Hochheiser (2017)).

We created a tutorial video to fill this background gap. The video is an extended and detailed version of the contents presented from Section 1.1 to Section 1.2. This video has a duration of about 40 minutes and was presented to each participant right before the start of his/her tasks[7].

We performed our study in a research lab devoted to user studies at our institution, where a participant is alone in the experiment room and the experimenter is at an adjacent observation room overseeing the experiment. This arrangement guaranteed a stable and regular environment without any distractions, with controlled factors such as lighting of the room and position of the participants in relation to the screen, keyboard and eye-tracker. For our study, we used the Tobii Pro Fusion bar, see Figure 1.5 eye-tracker and Tobii Pro Lab see Figure 1.6 tool for capturing and analyzing the data. The session of each participant followed the sequence:

- Brief introduction of the experiment by the experimenter, describing its goal and the protocol to follow.
- Signing the consent form in accordance with the ethics certificate of our institution [8].
- Watching training video and clarifying any questions or issues that the participants may have.
- Calibration of the eye-tracker using Tobii Pro Lab tool.

[7] The video cannot be anonymized, but will be made available as part of our replication package.

[8] Comité d'éthique de la recherche École de technologie supérieure: H20210602

- Warm-up practice with two questions in the Java application (see Section 3.3.2) to gain familiarity with the graphical interface.
- Semi-structured interview to gather further insights from the participants. The interviews were recorded for subsequent analysis.

More formally, our empirical study is a factorial design with two factors that correspond to our independent variables (Wohlin, Runeson, Höst, Ohlsson & Regnell (2012)). The first factor is a Number of Features (NoF) with 4 categorical values, and the second factor is a Number of Cross-Tree Constraints (NoC) with 3 categorical values, respectively corresponding to their number of ranges as explained in Section 3.3.

Recall that our Java interface collects two pieces of information for each question: i) correctness, with values `true` and `false` to indicate whether the response was correct or not, and b) response time in milliseconds, i.e. the elapsed time between clicking on the `Start` button and hitting the `Submit Answer` button. Thus, these are our first two dependent variables, respectively, in nominal and ratio scales. They are used for answering our first two research questions, RQ1 and RQ2.

We used a within-subject design, therefore, each participant answered all 24 comprehension questions. There is no time limit for answering tasks. This allows participants to begin the experiment and continue as they are ready. In addition, they can take a break between tasks, so long as they stay in their position and do not do anything requiring another calibration process.

With the aid of Tobii Pro Lab, we defined several areas of interest (AOIs) on top of the Java interface, see Figure 3.1. The AOIs roughly correspond to the elements in the graphical interface: ① feature model (FM), ② legend label (Legend), ③cross-tree constraints area (CTC), ④ question panel (Question), ⑤ answer panel (Answer), ⑥ buttons panel (Buttons), and ⑦ window. We should note that the AOI that corresponds to the feature model, i.e. ① in the figure, is tailored to each of the 24 feature models we used in our study. Similarly, the area of interest of CTCs, ③ in the figure, was removed when there were no CTCs and adapted according to the number of CTCs of the feature model. The rest of the graphical elements were fixed on the same area of the interface across all questions. For all our AOIs, we collected the fixations count and the fixation time on them. We normalized these values as percentages to analyze across questions and participants Holmqvist & Andersson (2017). We used these data for the research questions RQ3 and RQ4.

3.3.4 Participants

We recruited 19 participants, two of them piloted our experimental design, mostly graduate students from our institution and without prior knowledge of feature models.

For recruitment we considered two basic exclusion and inclusion criterion. We excluded participants with vision impairment and we included only software engineering that had basic knowledge about programming.

3.3.5 Experiment Execution

The experiment was conducted in November 2021 in an experiment room considered for eye-tracking studies. The room is equipped with a 29" screen, with 1920×1080 resolution, a mouse, a screen-based eye-tracker installed on the monitor used by participants, see Figure 3.3, an ergonomic chair and a laptop to run the experiment controlled by the experimenter. First, we provided a brief explanation of the experiment steps. Then, participants watched a 40-minute educational video. Next, before the calibration process, they can discuss issues and questions they faced while watching the video. In the next step, the experimenter calibrates the eye-tracker with the participant's position according to Tobii Pro Fusion guidelines. During the experiment, participants were asked to not move around. They can only rest their heads and eyes. The experiment takes between 20 and 25 minutes, depending on the participants' performance. There were no time limits, so they were free to spend as much time as they needed to find answers. We observed participants from the observational room, so during the experiment participants were alone in the room. As the last step, we interview participants for 10 to 15 minutes to find out

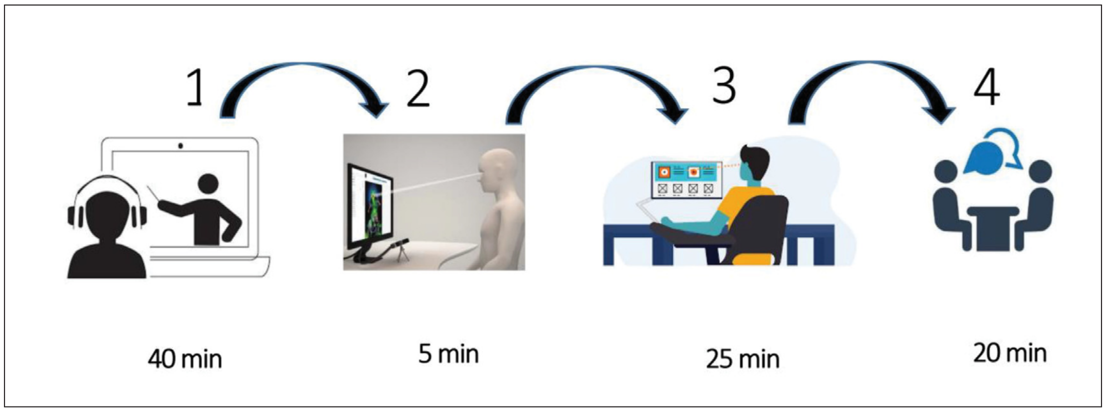


Figure 3.2 The process of experiment execution – (1) Educational video, (2) Calibration process, (3) Performing Tasks, (4) Interview

more about their opinions, suggestions, and difficulties they may have. We record their voice for further documentation, See Figure 3.2

3.3.6 Eye-Tracker Specification and Setup

In this experiment, we employed Tobii Pro fusion to collect participants gaze points. This eye-tracker has 120 Hz sampling frequency with 0.3 accuracy at optimal conditions. Current version supports both dark and bright pupil tracking. The data sample output of eye-tracker are Timestamps, Gaze origin, Gaze point, and Pupil diameter.

To calibrate participant's eyes, we followed the device recommendations provided by tobii Pro[9]. The distance between the participants and the monitor was set between 60 cm and 65 cm, for which the freedom of head movement is $40cm \times 25cm$ $(width \times height)$. We chose 9 dots calibration mode, to be followed by participants on the screen, which is shown in Figure 3.4. Before the calibration process, participants were asked to seat in a position they feel comfortable and not to move while doing the calibration. For any observed errors and inaccuracy, we recalibrate again.

[9] https://www.tobiipro.com/product-listing/fusion/#Specifications

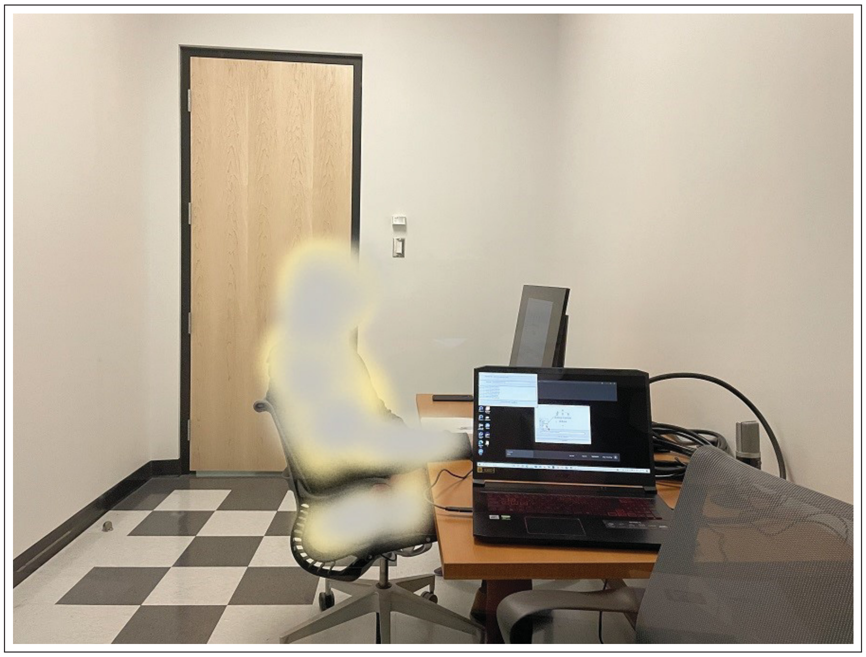
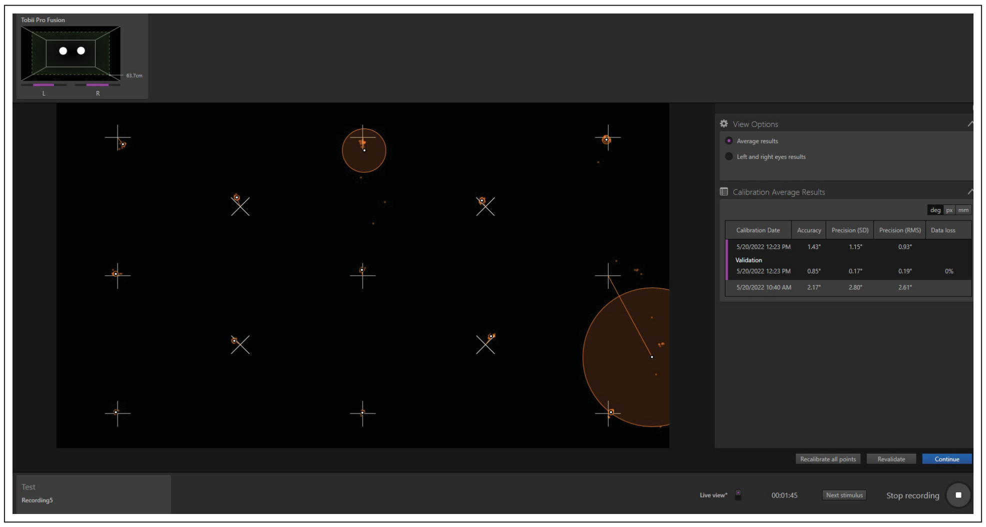


Figure 3.3 Experimental Room Layout

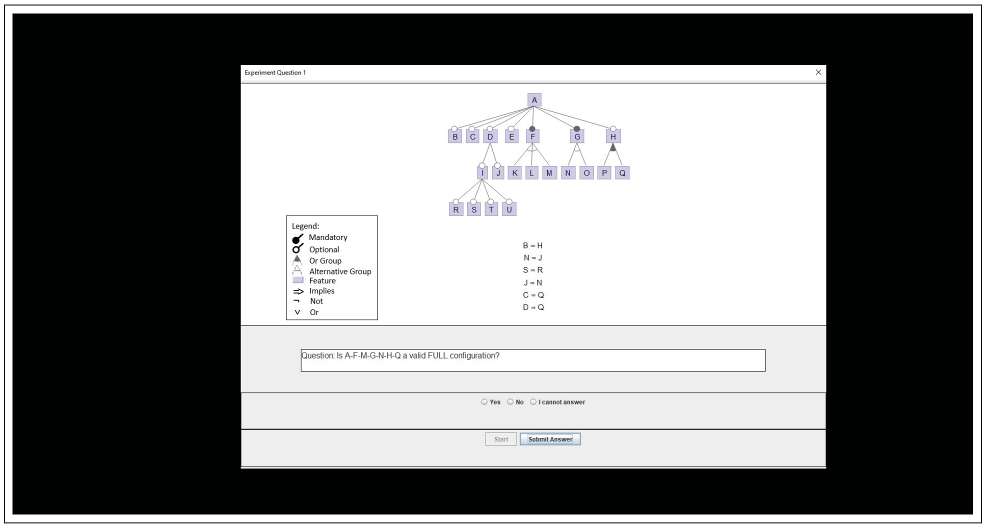


[✓ Average results]
Left and right eyes results

Calibration Date	Accuracy	Precision (SD)	Precision (RMS)	Data loss
5/20/2022 12:23 PM	1.43°	1.15°	0.93°	
Validation				
5/20/2022 12:23 PM	0.85°	0.17°	0.19°	0%
5/20/2022 10:40 AM	2.17°	2.80°	2.61°	

Figure 3.4 9 dots Calibration Process

We set the black background of the screen and place the experiment's window in the center of the screen, see Figure 3.5. Throughout the experiment, participants only need the mouse to start and submit each of their answers.



Question: Is A-F-M-G-N-H-Q a valid FULL configuration?

Figure 3.5 Task window

3.4 Data Extraction

We adapted the following steps to extract participants' raw data from Tobii Pro Lab:

1. For each 24 feature models, we defined its AOIs based on the goal of the study. These 24 unique patterns are baselines to designate AOIs of all participants performing 24 comprehension tasks and extract fixation and saccade related information for each question and each participant in Excel format.
2. Using recorded video via Tobii Pro Lab for each participant, we specified the duration of each question by defining start and end events for all tasks. In other words, we customized Times of Interest (ToI) for each task. See Figure 3.6. With the Tobii Pro software, we can define TOIs for whole tasks of each participant separately and be able to have all data related to each task in a Excel file, such as fixation count, fixation duration, spent time, and many other data that can be put in a final Excel file.
3. We saved the frame of each task as media (all frames are listed in Media Selection tab of Tobii Pro Lab) and set them for defined ToI of same task. We did the same process for all 24 tasks of each participant. See Figure 3.7. Considering that the order of the tasks different for each participant, therefore, we specified the actual number of questions beside the current order in naming process of defining ToIs and frames. This helped us to have

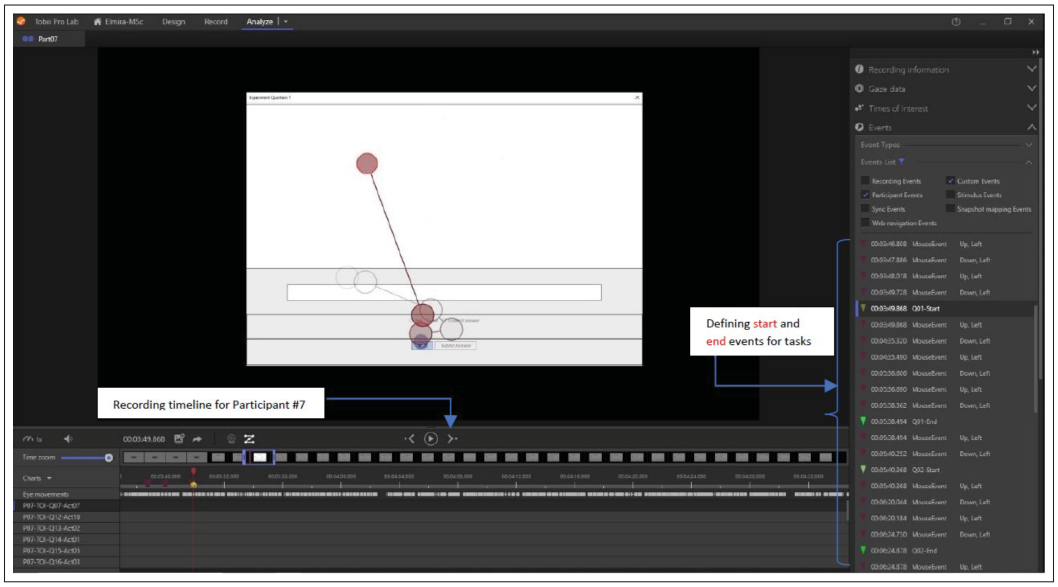

Defining start and end events for tasks
Recording timeline for Participant #7

Figure 3.6 Time of Interest (ToI) specified with Tobii Pro Lab

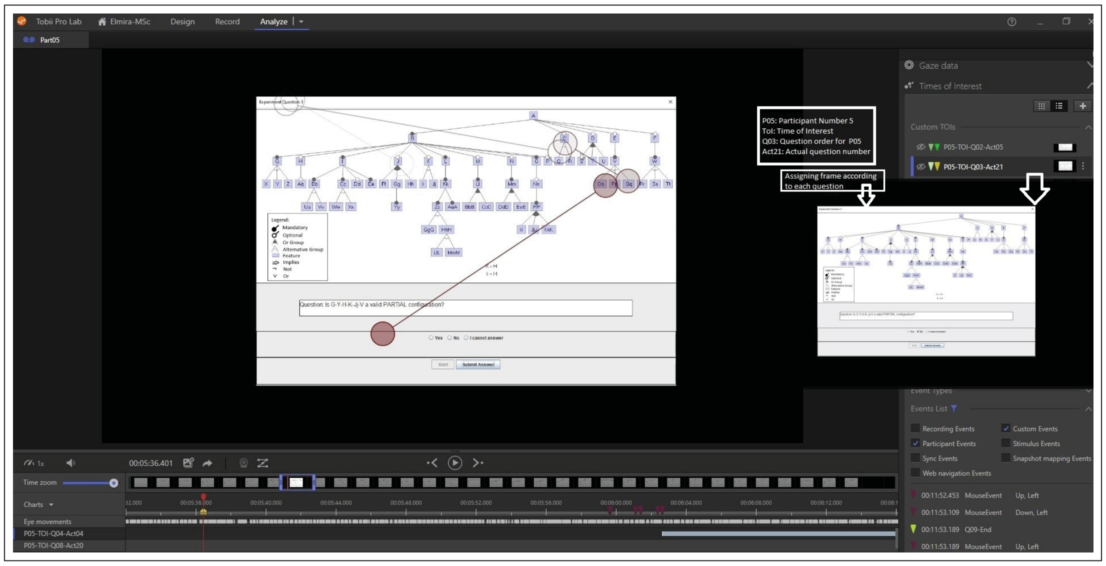

P05: Participant Number 5
ToI: Time of Interest
Q03: Question order for P05
Act21: Actual question number
Assigning frame according to each question

Figure 3.7 Naming process and assigning frame to ToI

exact and correct comparison of data among participants in the output file. Saving the frame of tasks also will be used in next steps for drawing heatmaps and defining AoIs for tasks of each participant.

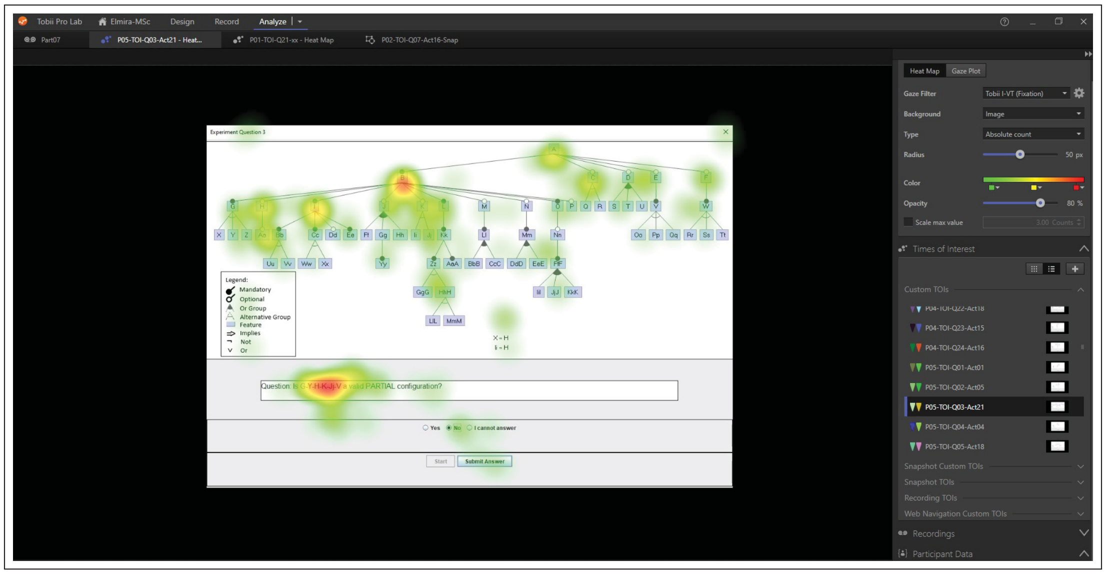


Figure 3.8 Creating heatmap for each task

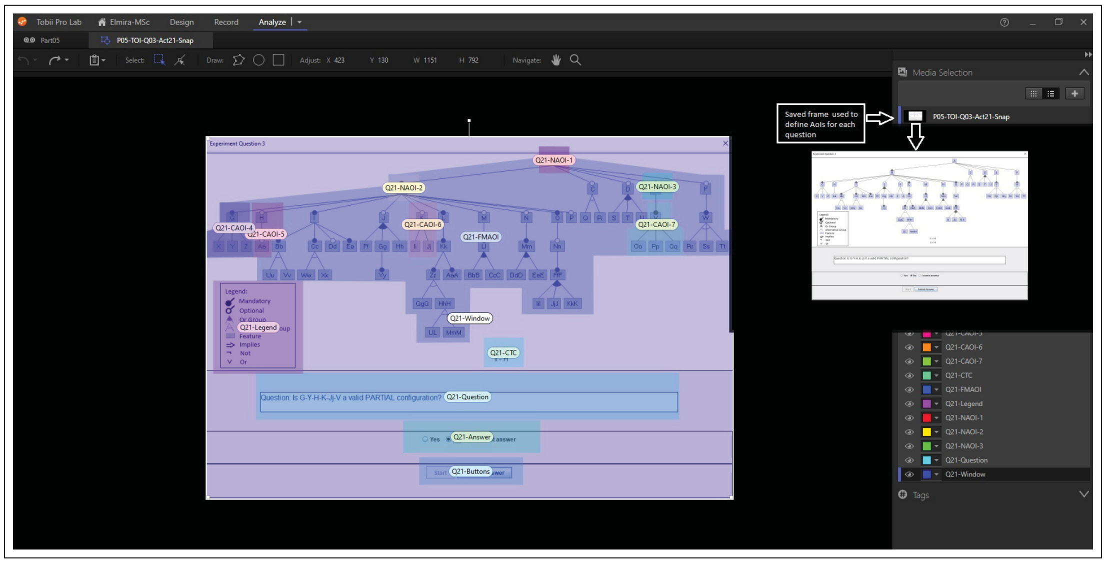


Figure 3.9 Assigning AoIs for each task

4. We used one of the visualization technique (heat map) provided by Tobii Pro Lab in order to assign AoIs to different regions of 24 feature models. See Figure 3.8. For each participant we saved heat Maps of 24 tasks and used them to specify the exact place of AoIs while adapting the standard format of AoIs explained in the first step above. See Figure 3.9.

5. The final step is to export data for each participant by specifying the number of questions, so we had 24 Excel files for each participant. By exporting the results of each question in a separate Excel file, we can compare the data of different participants for the same question. Within the *Data Export* window, we can specify the participant number as well as which task's data to include from the *Data selection* menu. We can select from the same window data types such as fixation, general participant information, gaze events, and others. The final output file will contain all these data.

3.5 Data processing and analysis plan

When quantitative research is conducted, the first step in statistics is to describe characteristics of the responses, and visualize how the data distributed. Considering the nature of data which can be nominal, ordinal, interval (discrete), and ratio (continuous), we can apply two basic calculation: measuring the central tendency (mean, mode, and median) and variability or spread (standard deviation, variance). In our study we also first applied descriptive statistics for correct and incorrect answers, response time for correct answers, and average proportions of fixation time and count for designed AoIs. We elaborate on this in Section 4.1. All descriptive statistics are done in R and the implementation is included in a replication package accessible from Scholar Portal Dataverse[10].

Following this is inferential statistics, which tell us whether the data confirm our Null hypothesis. To apply proper inferential statistical method, we looked at the nature of our research questions as well as the number of independent and dependent variables. In our first research question, the goal is to predict a group membership, and we analyzed the effect of two independent variables on one dependent variable, See Section 4.2. For the second research question and its subset, we also followed the same strategy and looked at our number of variables and the goal of the research question, which is finding the significance of group differences, See section 4.3. Inferential Statistics were processed with SPSS tool, and no script available for this part.

[10] https://doi.org/10.5683/SP3/HUOVO3

The diagrams below show our process for obtaining our data, see Figure 3.10. In green are the public elements that we have developed (images, code, etc.). In red is the information that we will keep private because of confidentiality as requested by the Ethics Committee (demographic information and raw data from the eye tracker during the execution of the tasks).Uncolored boxes represent the parts of the experiment we performed manually.

The second figure shows the process of data aggregation, see Figure 3.11. For each participant, we generated a program that extracts from the raw data the relevant information for our experiment. Finally, the performance information and the gaze of each participant are aggregated in a file that contains all the information of our experience. We categorized the elements of this process with respect to their availability, relevance for the replication of our work, or how they were performed. The categories indicated by different colors are:

1. Open: These artifacts are the key elements for replicating and reproducing our results. They are contained in the link for their evaluation.
2. Closed: These artifacts implement auxiliary and simple tasks that can be implemented in different ways and are not essential for replicating and reproducing our results. They are not contained in the link.
3. Personal: These are data files with personal or raw data from the experiments. They contain sensitive non- anonymized information. Our ethics certificate does not allow us to share this information.
4. Proprietary: This element is a proprietary tool for which a license must be paid.
5. Manual: Indicates a manual activity that was not automated. The first three such activities relate to the production of the image files of the feature models that we used as stimuli. If these activities are performed by a different person, the outcome would be essentially the same because the procedures are straightforward, so these activities are reproducible. However, for Eye-Tracker Data Curation, the outcome could be slightly different because this activity requires applying personal judgment to mark the AOIs for the eye-tracker data, which can vary slightly from participant to participant. We established some guidelines for this activity and two team members performed it.

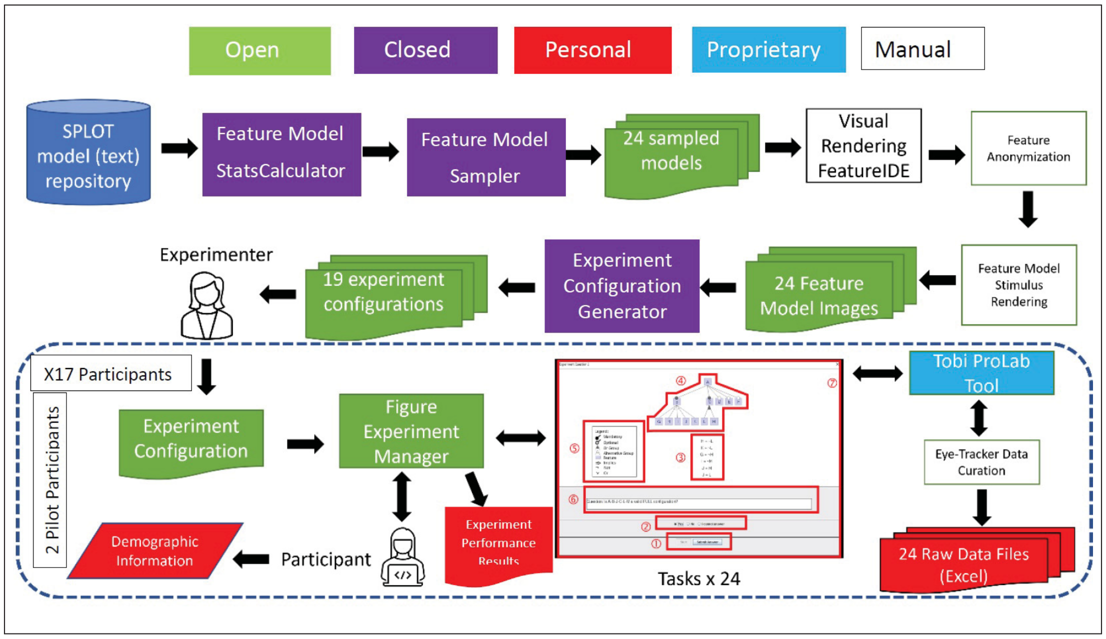

Open
Closed
Personal
Proprietary
Manual
SPLOT model (text) repository
Feature Model StatsCalculator
Feature Model Sampler
24 sampled models
Visual Rendering FeatureIDE
Feature Anonymization
Feature Model Stimulus Rendering
24 Feature Model Images
Experiment Configuration Generator
19 experiment configurations
Experimenter
X17 Participants
2 Pilot Participants
Experiment Configuration
Figure Experiment Manager
Experiment Performance Results
Demographic Information
Participant
Tasks x 24
Tobi ProLab Tool
Eye-Tracker Data Curation
24 Raw Data Files (Excel)

Figure 3.10 Diagram shows the steps of experiment from SPLOT repository until data extraction – Green: Public sections, Red: Private sections, Purple: research-based sections, Uncolored: Not automated sections, Blue: Eye-tracker software

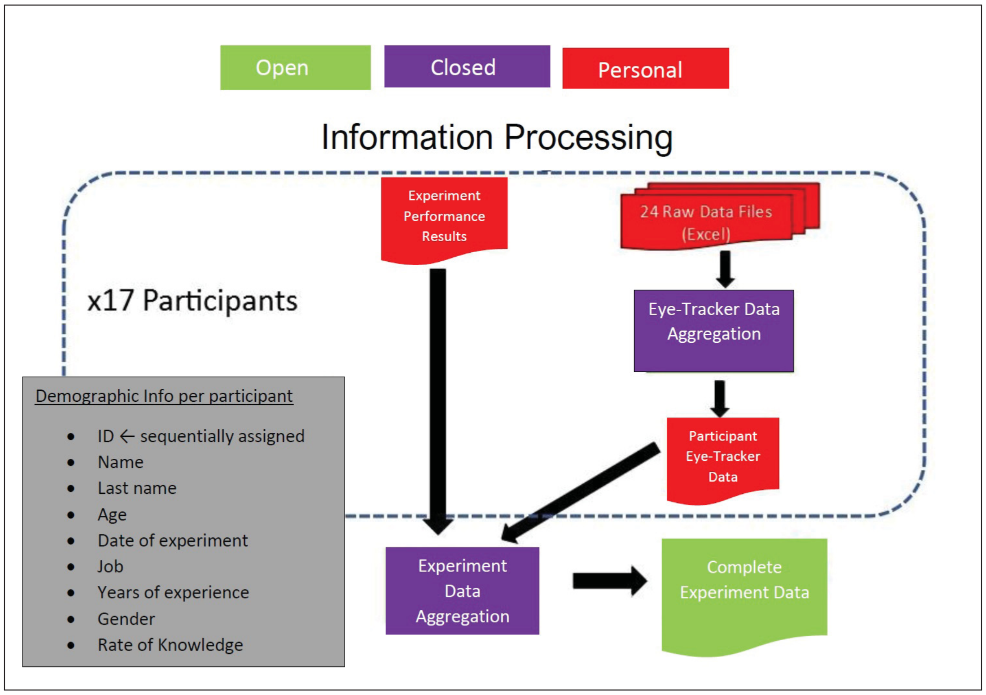

Open
Closed
Personal
Information Processing
x17 Participants
Experiment Performance Results
24 Raw Data Files (Excel)
Eye-Tracker Data Aggregation
Participant Eye-Tracker Data
Demographic Info per participant
ID ← sequentially assigned
Name
Last name
Age
Date of experiment
Job
Years of experience
Gender
Rate of Knowledge
Experiment Data Aggregation
Complete Experiment Data

Figure 3.11 Diagram shows the steps of data collection for each participant

3.5.1 Interviews – Qualitative analysis

We conducted semi-structured interviews at the end of each experiment. As part of the interviewing process, we recorded and manually transcribed the voices of participants. The interviews started with warm up questions, such as their opinion about SPL and eye-tracking technology. Then, we continued with the main questions, starting with series related to educational video and their feedbacks. The goal was to find out if they could answer the questions relying on contents presented in the video, and if the given information relevant to the designed questions. As a next step, we asked about the difficulty of the tasks, including which ones were difficult to complete and why. Afterwards, we asked about the strategy they used to solve the problems and what they thought about the sequencing of the tasks. We finished the conversations by receiving their opinions and comments about their overall experience about the experiment.

It is assumed that participants' answers in the interview sessions asking about the difficulty of tasks and the strategy they used to find the answers will be consistent with data collected on their eyes' movements. Following the proper statistical analysis, we expect to observe consistency between the answers derived from log files, eye-tracking data, and the responses they provided during interviews. The interpretation of interviews are explained in Section 4.4

3.6 Summary

In this chapter, we explained how we designed and conducted our experiment. We introduced and discussed our research questions, the rationale behind each of them, the steps of derivation of feature models, recruitment of participants, the steps of tasks and stimuli selection, the installation of the eye-tracking device for the purpose of our study and how to obtain the participant data from its software (Pro Lab), then we briefly described the statistical analyses performed and at the end we summarized some key points from interview session.

CHAPTER 4

RESULTS AND DISCUSSION

In this section, we start with a general overview of the results obtained before delving into the detailed analysis for each of our four research questions.

4.1 Descriptive Statistics

All participants completed the 24 questions, and only one participant in one question indicated that he/she could not answer the question. In total, for all participants, there were 284 correct and 124 incorrect answers. Participants had between 9 and 21 correct answers, with a median of 17, and conversely they had between 3 and 15 incorrect answers, with a median of 7. On average, the correct answers took 42.84 seconds to respond, with a standard deviation of 26.15 seconds. For the incorrect answers, the average response time was 52.96 seconds, with a standard deviation of 37.82 seconds. Figure 4.1 summarizes the distribution of correct and incorrect responses across the ranges of NoF and NoC metrics described in Section 3.3. Notice that in eleven out of twelve combinations of ranges, the number of correct answers is higher than the number of incorrect answers. The only exception is when the number of features is between 20 and 34 features, NoF range (3), with zero CTC, NoC range (1). In this case, the number of incorrect responses is slightly higher, with 18 versus 16 correct responses.

Figure 4.2 presents the distribution of response time in seconds for the tasks that were answered correctly. In this figure, it is important to highlight two patterns. The first pattern is that, in general, at each range of NoC (i.e. row-wise), the median values present a steady increase as the range of NoF increases. A similar second pattern occurs across the ranges of NoF (i.e. column-wise), that is, the median values also present an increase as NoC range increases.

Figure 4.3 shows the average of the proportions of fixation time and fixation count across the AOIs. The graphs show a clear pattern where there are three predominant AOIs, both in fixation time and count. The respective percentages for tasks with and without CTCs are as follows: *i)*

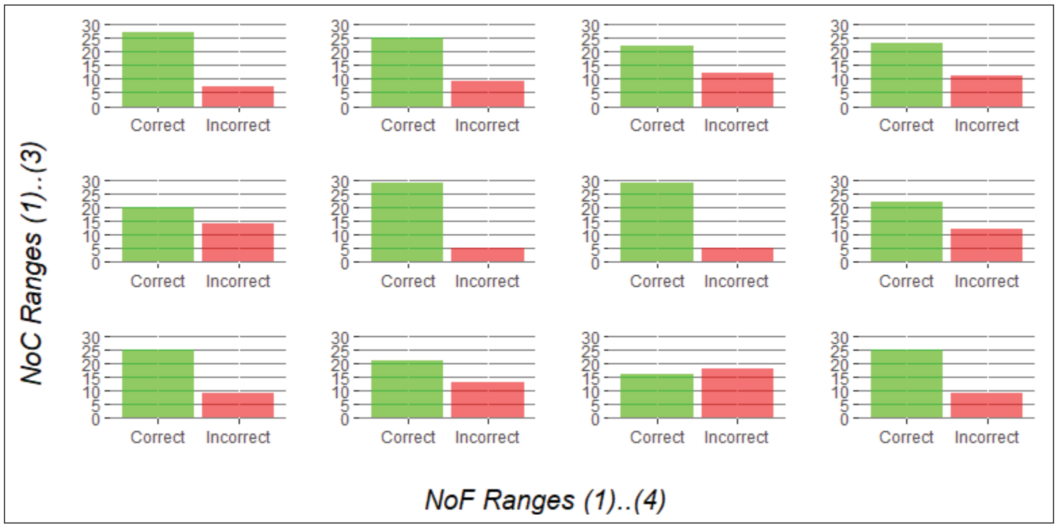


Figure 4.1 Correct and Incorrect Responses per NoF and NoC ranges

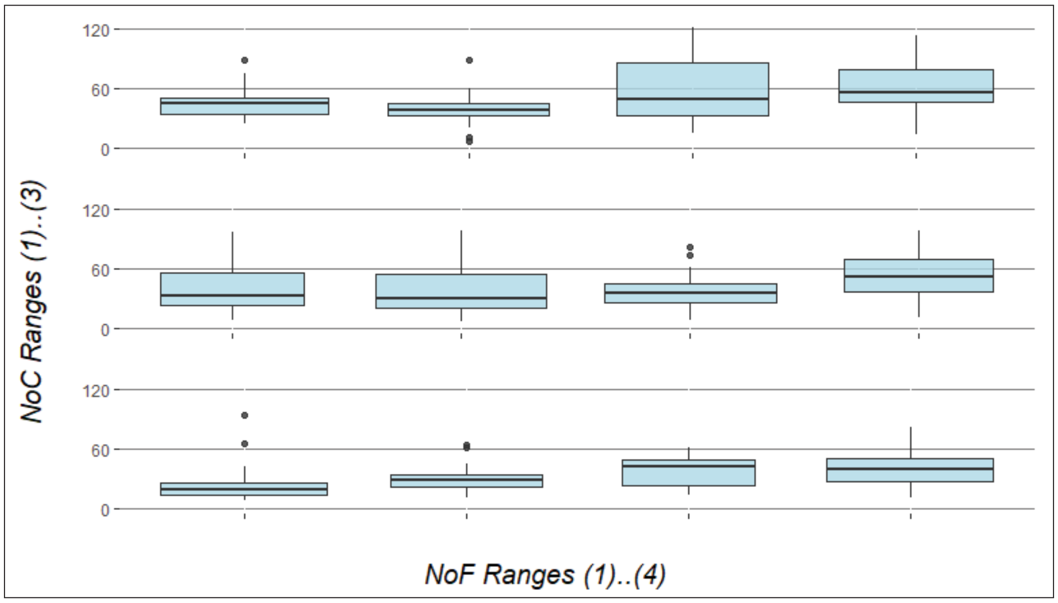


Figure 4.2 Response Time in seconds for Correct Answers per NoF and NoC ranges

FM for the feature model with fixation time of 41% and 53%, and fixation count 40% and 52%; *ii)* Question for the question panel with fixation time of 32% and 27%, and fixation count of 33% and 29%; *iii)* CTC for the cross-tree constraints with fixation time of 11% and fixation count of 11%. These findings support our postulate in Section 1.3.7 that a cognitive model for feature model comprehension contains three mental models that relate to each of the predominant AOIs. Furthermore, the percentage distribution suggests that the cognitive effort devoted to the CTCs is instead shifted to the effort related to FM when the tasks do not involve CTCs.

4.2 RQ1 results

We discuss now the results of the two research questions derived from RQ1.

Results of RQ1a. Recall from Section 3.1 that this question aims at identifying any effects of NoF and NoC on task accuracy. Hence, the independent variables are the NoF and NoC that are of categorical scale, and the dependent variable is task accuracy (with two levels correct and incorrect).

To answer part *a* of the first research question, we must check the possibility of any effect that might each of independent variables or the joint of two have on the dependent variable. For this reason, we adapted a statistical method that best fits to our variables' conditions. Referring to Tabachnick & Fidell (2013), the best statistical technique that fulfills the goal of question and satisfy the variables specifications is multiway frequency analysis. This is a nonparametric

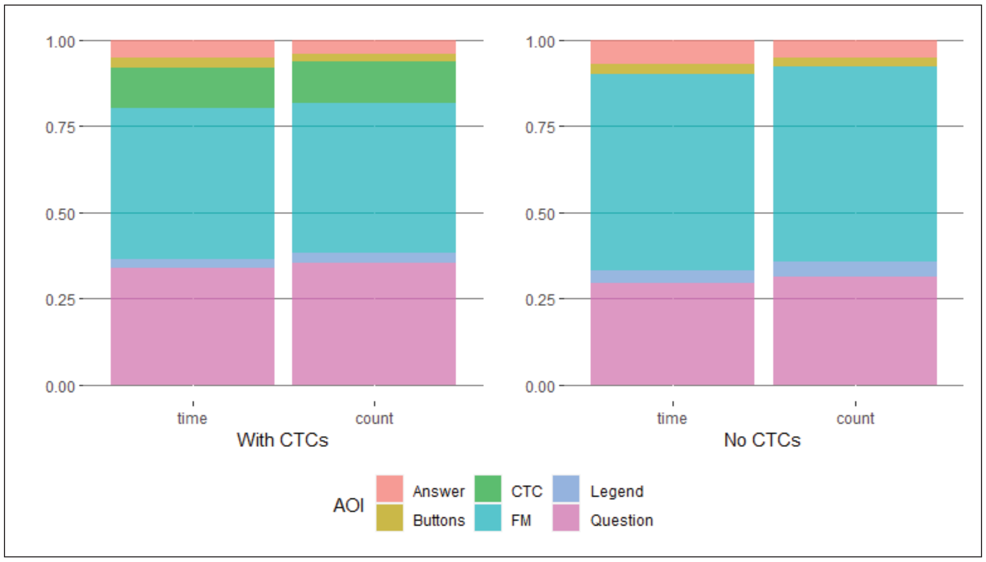


Figure 4.3 Average Proportions of Fixation Time and Count

method used to analyse categorical and qualitative discrete dependent and independent variables. In this test, which is a generalization of the chi-square, the frequency of a cell (i.e., a combination of factors) is the dependent variable that is influenced by one or more categorical factors and their associations Tabachnick & Fidell (2013). In this method, the null hypothesis H_0 is that the factors do not affect the frequency distribution of a dependent variable.

Our results show that there are neither main effects of NoF nor of NoC on task accuracy. However, there is an interaction effect of NoF and NoC on task accuracy ($\chi^2_6 = 21.82$, $p < .001$). The higher the value of NoC caused a lower task accuracy, but this drop is different across the values of NoF, which also contributes to lowering task accuracy. In other words, NoF and NoC are additive in causing a lower task accuracy. This result suggests that the mistakes in the responses, attributable to comprehension breakdowns, more likely come from the interactions that must occur among the mental models, as they are constructed and manipulated to perform the tasks, rather than the values NoF or NoC by themselves.

Results of RQ1b. Recall from Section 3.1 that this question aims at identifying any effects of NoF and NoC on the response time of the tasks that were answered correctly.

We adapted a two-way ANOVA test to meet the requirements of a research question in which we are seeking the main effect as well as the interaction of two categorical independent variables (NoF and NoC) called factors on a continuous dependent variable (response time). As factors can be divided into levels, the two independent variables (NoF and NoC) in our experiment each have four and three levels respectively; we, therefore, examined the effect of each level for all possible combinations of factors (3 x 4) on increasing the response time for the given correct answers.

Each factor effect is examined separately, and for interaction effect both factors are considered at the same time. The ANOVA table has 12 cells considering levels of two factors (NoF=4 x NoC=3) and in each cell we calculate mean spent time of correct answers for our 17 participants.

There are three sets of Hypothesis for this research question:

- H01: The means of each row (rows represent NoF levels) are equal.
- H02: The means of each column (columns represent NoC levels) are equal.
- H03: There is no interaction between the row (NoF) and column (NoC).

In other words, the null hypothesis H_0 is that the means of the dependent variable across both factors and their interaction are equal Tabachnick & Fidell (2013).

For correct answers, both NoF ($F_{3,272} = 8.28$, $p < 0.001$, $\eta_p^2 = 0.084$) and NoC ($F_{2,272} = 11.58$, $p < 0.001$, $\eta_p^2 = 0.078$) have a main effect on response time, but there is no interaction effect of NoF and NoC on response time. NoF causes an almost steady increase in time needed to answer correctly, and the same is true for NoC. This result indicates an augmentation in the cognitive processing needed to solve tasks. Such augmentation is in line with the limitations of the working memory and reflect the need to (re)assemble the necessary mental models, or parts thereof, in order to perform the necessary operations with them to successfully complete the tasks.

This result also raises the question of which part(s) of the visual stimuli of the tasks need to be processed. To answer this question, we will examine closely the visual attention allocated to the different AOIs in the next research question.

4.3 RQ2 results

We discuss now the results of the two research questions derived from RQ2.

Results of RQ2a. Recall from Section 3.1, that here we want to find out if the ratio of fixation time of any AOI can be used to explain the accuracy of the responses. As result, in this research question, the ratio of response time of all defined AOIs are dependent variables, therefore, we have multiple continuous dependent variables corresponding to the number of AOIs of stimuli and two discrete independent variables, tasks with and without NoC areas. Considering all these specifications and the goal of the research question, we applied factorial MANOVA statistical test. We believe that the given correct answers for tasks with presence and absence of NoC affect the ratio of response time in each AOIs. Therefore, we applied MANOVA test twice, once for stimuli without CTC and its defined AOI and second time applying the test for same stimuli with presence of NoC area (AOI of CTC) Tabachnick & Fidell (2013).

The only difference between ANOVA and MANOVA tests is the number of dependent variables, and because in this research question we have more than one dependent variable, therefore we applied MANOVA test.

For tasks with CTC, the proportion of time fixating each AOI is related to the accuracy of answers ($F_{6,251} = 3.027$, $p < .007$, $\eta_p^2 = .067$). Additionally, there is no interaction effect of accuracy of answers with the experimental factors, NoF and NoC. Regarding the relation between accuracy of answers and each of the specific dependent variables (AOIs), the tests are significant for: FM ($F_{1,256} = 15.952$, $p < .001$, $\eta_p^2 = .059$), Question ($F_{1,256} = 16.486$, $p < .001$, $\eta_p^2 = .061$), and answer ($F_{1,256} = 4.796$, $p < .029$, $\eta_p^2 = .018$). Differences in the means of fixation time ratios for given AOIs for correct vs incorrect responses were as follows: FM 40% vs 49%, Question 34% vs 28%, and Answer 6% vs 4%. This finding means that devoting more cognitive effort in comprehending the FM AOI is related to a higher likelihood of providing an incorrect answer. Inversely, devoting more cognitive effort to understanding the AOIs Question or Answer is related to a higher likelihood of providing a correct answer. Contrary to RQ2b, the proportion of fixation time for the CTC is not significant. This suggests that the impact of the CTC on cognitive overload is not a matter of time, but rather how often it is the object of the visual attention.

For tasks with no CTC, the proportion of time fixating each AOI is not related to the accuracy of answers ($F_{5,124} = 0.197$, $p > .05$). Since the null hypothesis for the omnibus test could not be rejected, no further statistical analysis was required.

Results of RQ2b. Recall from Section 3.1, that for this question we want to find out if the ratio of fixation counts of the AOIs can be used to explain the accuracy of the responses.

Consequently, the analysis focuses primarily on the accuracy of answers, with the eventual examination of interaction effects of this factor with NoF and NoC. Thus, accuracy, NoF and NoC are the independent variables and the ratios of fixation counts of the six AOIs are our dependent variables, that is, the AOIs Answer, Buttons, CTC, FM, Legend and Question as explained in Section 3.3.3[1]. Again, because tasks with CTCs involve the additional AOI CTC, we applied the MANOVA test twice, once for the tasks with CTCs and once for the tasks without CTCs Tabachnick & Fidell (2013).

For tasks with CTCs, the MANOVA test considered the factors (2:accuracy x 4:NoF x 3:NoC). For this test, the null hypothesis H_0 for the omnibus test states that the factors are not related to the variance of the dependent variables taken globally. When the omnibus test is significant for one factor or a combination of factors, the null hypothesis H_0 for each dependent variable is that this factor, or combination of factors, is not related to the variance of that dependent variable. We found from the omnibus test that the accuracy of answers was significant globally, that is, the ratio of fixation counts of each AOI is related to the accuracy of answers ($F_{6,251} = 3.458$, $p < .003$, $\eta_p^2 = .076$), with no interaction effect of this factor with NoF and NoC. Because of this finding, we analyzed the implication of accuracy on each of the six dependent variables. We observed significant differences in the means of the following AOIs: FM ($F_{1,256} = 19.150$, $p < .001$, $\eta_p^2 = .070$), CTC ($F_{1,256} = 8.088$, $p < .005$, $\eta_p^2 = .011$), Question ($F_{1,256} = 17.723$, $p < .001$, $\eta_p^2 = .065$), and buttons ($F_{1,256} = 4.032$, $p < .046$, $\eta_p^2 = .016$). Differences in the means of fixation count ratios for given AOIs for correct vs incorrect responses were as follows: FM 39% vs 48%, CTC 12% vs 9%, Question 35% vs 28%, and Buttons 2% vs 1.7%. This finding means that devoting more cognitive effort in constructing and manipulating the mental model of the feature model is related to a higher likelihood of providing an incorrect answer. Inversely, devoting more cognitive effort to constructing and manipulating the mental models of the CTCs or of the configuration, and looking more at the Buttons AOI, is related to a higher likelihood of providing a correct answer.

[1] Notice that the Window AOI is not included in the analysis because it was used only as a fallback area for fixations outside the other six AOIs

For tasks without CTCs, the MANOVA test considered the factors (2: accuracy x4: NoF) because the value of CTC is now a constant. The results show that the null hypothesis of the omnibus test could not be rejected ($F_{5,124} = 0.296$, $p > .05$), therefore the results for each of the dependent variables were not examined. In other words, the proportion of fixation counts of each AOI is not related to the accuracy of answers.

Recall that incorrect answers are theoretically attributed to comprehension breakdowns linked to cognitive overload. The observation of the association of a higher proportion of cognitive effort devoted to the mental model of the feature model with the higher chance of an incorrect answer, can be attributed to the fact that the feature model contains many more items than the working memory can simultaneously hold, and it cannot be chunked (i.e. broken down) into manageable units easily. In addition, this pattern in visual attention may be related to an inefficient strategy of trying to memorize (i.e., build a complete mental model) the full feature model, which is not possible. Inversely, a higher proportion of cognitive effort on the other AOIs is possibly related to a strategy of staying within the limits of the working memory by focusing on one item of the Question AOI in relation to the AOIs FM and CTC as needed. For tasks without CTCs, the absence of associations between cognitive effort about the various AOIs and accuracy suggests that such tasks are less likely to overload the working memory, irrespective of the amount of cognitive effort devoted to the various aspects of the tasks. This is in line with the results for RQ1a, which concluded that both NoF and NoC were necessary to have an impact on accuracy.

These results converge with those from RQ2a, and reinforce the previous interpretation. However, the role of the CTC needs to be further nuanced by referring to the difference between fixation time and fixation count. As indicated before, the fixation time on an AOI is the most global indicator of the difficulty of constructing the mental model associated with the corresponding visual component. In contrast, fixation count is indicative of the effort devoted to understanding the various components of the visual stimulus, and it reflects more directly the different pieces of visual information processed as NoF and NoC vary. Hence, because the CTC AOI contains a series of relatively simple logic formulas, it is plausible that the results of RQ2b in terms of fixation counts reflect a process of information intake whereby formulas in the CTC are fixated on as needed but with a small impact on fixation time to become relevant in the overall time of the construction of the mental models required to successfully complete the tasks.

4.4 Interview Results

In this section we present the result of conducted semi-structured interview and highlight some key questions along with given answers. These finding support our hypotheses and assumptions about three predominant area of interests (AOIs) and are consistent with results of our statistical analyses for RQ1 and RQ2. Participants stated that they mostly focused and spent more time on FM, CTC and Question AOIs which also align with our findings. They conveyed that for the tasks in which feature models were bigger in size and had more number of cross-tree constraints they spent more time to find answers regardless of the given answers were correct or not.

Did you find any question harder/more complex than others? Why? How so?

Participants reported having a difficult time checking the validity of feature models with too many features, even without CTCs. Whenever they encountered questions about full valid configuration, they found it difficult to locate features in the feature model and control all hierarchical constraints. The majority of participants find feature models with too many features and fewer CTCs difficult to answer, as opposed to feature models with fewer features and more CTCs. They argue that checking validity using CTCs is easier and more straightforward than checking hierarchy constraints. As such, to answer the question of feature models also containing CTCs, they first looked at whether CTCs contain any obligation; if not, they examined feature model constraints.

P03: Questions that contain cross-tree constraints were easier to answer than questions in which I only had to check hierarchical constraints inside diagrams. For me, answering questions asking about full valid configuration were easier because I can check the validity starts from top and go through the tree step by step.

P05: Questions were asking about validity of partial configuration in big diagrams, much harder for me to find answer. Generally, checking the validity in diagrams with too many features was harder compared to small tree with too many cross-tree constraints.

P07: I did not find any questions harder or more complex than others. Almost all questions had the same level of difficulty for me. However, the larger feature models with less CTCs were harder to find answers compare to small feature models with more CTCs.

P08: Big trees with less CTCs were harder to answer compared to feature models which were small but have too many CTCs. Finding features in the tree to check the validity are more complicated, and it requires more mental effort compared to feature models contain CTCs.

P12: Feature models, with too many features without CTCs, were complicated and checking FULL configuration was more challenging for me. Having CTCs in feature model helped me to check the validity faster.

P13: The ones with lots of CTCs and too many features were harder to answer. Feature models with too many features were more complicated to check the validity compared to small feature models with too many CTCs.

What patterns did you use to answer the questions?

Two groups of answers were found for this question. After watching the educational video, participants answered the question using the steps they learned. As they start with questions, they check the feature model constantly, locating letters and checking for constraints within the tree. As the last step in the process, they check CTCs. However, some participants altered their approach to finding answers after answering a few questions. First, for the features model with CTCs, they verified the CTCs' rules by finding letters in trees, and if they were unable to find any error in CTCs, they checked the feature model's constraints (hierarchical constraints).

P03: Look at letters inside the diagrams and then checking the list of constraints at the end. First, I am looking at feature models' constraints and then cross-tree constraints, based on what I learned from educational video.

P04: First. I saw letters in question then, check the feature models and at the end check the cross-tree constraints if there were any.

P06: For the feature models which were small, I checked the tree first. But, for big feature models, I tried to check in parallel the CTC and feature model together. Sometimes for big feature models, I was looking for exception through the CTCs first, because it is hard to check the validity of big feature models.

P08: It depends on questions. If the question had CTC, first I checked the CTC, then the feature model's constraints.

P10: First, I checked feature model then, CTC. However. In big diagrams for checking full valid configuration if it contains CTC, I was first looking for CTCs if they gave me clue to check invalidity if I could not find any then I check feature model's constraints.

P16: Starting with question, for feature models that contained CTCs, first I check CTCs then feature model.

P19: I started with feature model and relations between features then, I checked the CTCs.

While thinking about your responses, did you check the legend for the notation? If so, how often do you think it was?

We came to the conclusion that most participants barely checked the legend. According to their comments, they were able to remember all the information from the educational video as they watched it just before the experiment. Some participants had difficulty recalling X-Or and Or-group constraints, so they checked the legend while answering the first couple of questions.

P09: I did not check. I remembered all the details from the educational video.

P11: No, I did not check, because educational video explained clearly about these notations.

P06: I did not need to check the legend because I remembered all notations from the educational video.

P10: Yes, I checked the legend, at least for the first couple of questions.

P18: Sometimes, I have checked especially for X-or relation.

P13: Yes, few times.

4.5 Threats to Validity

Our empirical study faced several potential threats to validity as similar studies that rely on eye-tracker measures (Wohlin *et al.* (2012); Sharafi *et al.* (2020)). Concerning *internal validity*, a threat was *participant selection*. We addressed this issue by recruiting as large and diverse a set of participants as possible, both in terms of gender and technical knowledge of the subject. Because we used a within-subjects design, we faced an *order effect bias* whereby participants perform better as they performed more tasks. We mitigated this threat by using a random order of tasks for each participant. The *instrumentation* threat concerns the quality and reliability of the artifacts used in the experiment. We tackled this threat by following a rigorous protocol that included multiple checks from eye-tracker measures and calibrations for each participant to extensive consistency manual and automated validations of the collected and synthesized data.

Regarding *external validity*, a fundamental threat is the selection of the feature models. Certainly, selecting other data set of feature models might yield different results from ours. However, we argue that we addressed this threat by choosing feature models from the largest repository available and randomly across the twelve combinations of our two factors that are our feature metrics. Because the majority of our participants were graduate students without background in feature models, we cannot generalize our findings to other groups like experienced developers or variability experts. Nonetheless, this aspect is part of our future work. In terms of *construct validity*, we used common measures of cognitive load such as accuracy, response time, ratios of fixation counts and fixation time on AOIs which were analyzed with standard statistics procedures Holmqvist & Andersson (2017); Gonçales *et al.* (2021). Considering other eye-tracking and multimodal measures of cognitive load are also part of our future work.

CONCLUSION AND RECOMMENDATIONS

We presented the first empirical study on feature model comprehension using eye-tracking metrics for tasks of verifying validity in partial and full configurations. Our focus was on the impact of two feature model metrics, number of features (NoF) and number of cross-tree constraints (NoC), on these verification tasks. We found that these tasks are complex from a cognitive point of view because they are affected by multiple factors. For instance, tasks with more NoF or more NoC involved more cognitive operations to answer correctly, as shown by a steady increase in response time. Also, when considering the different aspects (AOIs) of the visual stimuli, our most striking finding was that looking more frequently or longer at the feature model (AOI FM) of a task increases the likelihood of providing an incorrect answer, irrespective of the NoC and NoF.

We proposed the foundations of a cognitive model for feature model comprehension, containing three distinctive mental models. Their relevance was attested by the collected fixation time and fixation count data obtained on their corresponding AOIs (FM, CTC, and Question). Building and manipulating mental models implies carrying out a sequence of steps or operations on them.

As future work, we plan to investigate how the transitions between fixations and across AOIs reflect these sequences. Our goal is identifying and characterizing the patterns or conditions that may lead to comprehension breakdowns. In addition, we will zoom in the FM AOI by breaking it into finer grain AOIs that would enable us to analyze model navigation strategies and their relation to mental model operations, and to study the impact of other feature model metrics such as branching factor. We also plan to consider other eye-tracking measures of cognitive load from pupil size to EEG Gonçales *et al.* (2021), and including a larger and more diverse participant population.

From participant's perspective, there are other important aspects of feature model that might affect feature model comprehension, such as depth of tree. Participants declared that those feature models in which features spread vertically were easy to follow the hierarchical constraints for them, comparing to horizontally distributed features in feature models. Accordingly, as feature work, we will investigate how layouts of feature models may facilitate their comprehension. Another important factor is the name of features, which in this experiment we decided to anonymize to prevent any bias in the task performance. Nevertheless, in the real world and for industry applications, it is critical to understand a feature model's name. Sometimes inappropriate labeling, such as selecting long and irrelevant names, makes understanding difficult. Therefore, study the effect of visual appearance of feature models by assessing the factors such as font, color, box-size of features and labeling is in our list to evaluate feature model comprehensibility.

We are just beginning to study feature model comprehension, and there are many metrics, See Table 2.4 that can be evaluated regarding their potential effect on understanding feature models. We will consider other structural metrics.

Ultimately, research on feature model comprehension will inform language design and tool development to provide more suitable language structures, user interfaces and support for this vital kind of SPL models.

BIBLIOGRAPHY

Apel, S., Batory, D. S., Kästner, C. & Saake, G. (2013). *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer. doi: 10.1007/978-3-642-37521-7.

Bagheri, E. & Gasevic, D. (2011). Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3), 579–612.

Batory, D. S., Sarvela, J. N. & Rauschmayer, A. (2004). Scaling Step-Wise Refinement. *IEEE Trans. Software Eng.*, 30(6), 355-371.

Benavides, D. (2019). Variability Modelling and Analysis During 30 Years. *From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*, 11865(Lecture Notes in Computer Science), 365–373. doi: 10.1007/978-3-030-30985-5_21.

Benavides, D., Segura, S. & Cortés, A. R. (2010). Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6), 615-636.

Bezerra, C. I., Andrade, R. & Monteiro, J. M. S. (2015). Measures for quality evaluation of feature models. *International Conference on Software Reuse*, pp. 282–297.

Bezerra, C. I., Monteiro, J. M., Andrade, R. M. & Rocha, L. S. (2016). Analyzing the feature models maintainability over their evolution process: An exploratory study. *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, pp. 17–24.

Bezerra, C. I., Andrade, R. M. & Monteiro, J. M. (2017). Exploring quality measures for the evaluation of feature models: a case study. *Journal of Systems and Software*, 131, 366–385.

Bidlake, L., Aubanel, E. & Voyer, D. (2020). Systematic literature review of empirical studies on mental representations of programs. *Journal of Systems and Software*, 165, 110565.

Busjahn, T., Schulte, C. & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, (Koli Calling '11), 1–9. doi: 10.1145/2094131.2094133.

Cagiltay, N. E., Tokdemir, G., Kilic, O. & Topalli, D. (2013). Performing and analyzing non-formal inspections of entity relationship diagram (ERD). *Journal of Systems and Software*, 86(8), 2184–2195.

Capilla, R., Bosch, J. & Kang, K. C. (Eds.). (2013). *Systems and Software Variability Management - Concepts, Tools and Experiences*. Springer. doi: 10.1007/978-3-642-36583-6.

Cárdenas-Figueroa, A. & Navarro, A. O. (2020). Overview of Mental Models research using bibliometric indicators. *Cognitive Processing*, 21(2), 155–165.

Cepeda Porras, G. & Guéhéneuc, Y.-G. (2010). An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering*, 15(5), 493–522.

Chen, L. & Ali Babar, M. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4), 344–362. doi: 10.1016/j.infsof.2010.12.006.

Chen, L. & Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Inform. & Software Tech.*, 53(4), 344-362.

Chen, O., Paas, F. & Sweller, J. (2021). Spacing and interleaving effects require distinct theoretical bases: a systematic review testing the cognitive load and discriminative-contrast hypotheses. *Educational Psychology Review*, 33(4), 1499–1522.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K. & Wasowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. *VaMoS*, pp. 173–182. doi: 10.1145/2110147.2110167.

da Costa, J., Gheyi, R., Ribeiro, M., Apel, S., Alves, V., Fonseca, B., Medeiros, F. & Garcia, A. (2021). Evaluating refactorings for disciplining #ifdef annotations: An eye tracking study with novices. *Empirical Software Engineering*, 26(5). doi: 10.1007/s10664-021-10002-8. cited By 0.

De Smet, B., Lempereur, L., Sharafi, Z., Guéhéneuc, Y.-G., Antoniol, G. & Habra, N. (2014). Taupe: Visualizing and analyzing eye-tracking data. *Science of computer programming*, 79, 260–278.

Détienne, F. (2001). *Software design–cognitive aspect*. Springer Science & Business Media.

Duarte, R. B., da Silveira, D. S., de Albuquerque Brito, V. & Lopes, C. S. (2020). A systematic literature review on the usage of eye-tracking in understanding process models. *Business Process Management Journal*.

Duchowski, A. T. (2017). *Eye Tracking Methodology - Theory and Practice, Third Edition*. Springer. doi: 10.1007/978-3-319-57883-5.

El-Sharkawy, S., Yamagishi-Eichler, N. & Schmid, K. (2019). Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Inf. Softw. Technol.*, 106, 1–30. doi: 10.1016/j.infsof.2018.08.015.

El-Sharkawy, S., Yamagishi-Eichler, N. & Schmid, K. (2019). Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology*, 106, 1–30.

Feigenspan, J., Kästner, C., Apel, S., Liebig, J., Schulze, M., Dachselt, R., Papendieck, M., Leich, T. & Saake, G. (2013). Do background colors improve program comprehension in the #ifdef hell? *Empir. Softw. Eng.*, 18(4), 699–745. doi: 10.1007/s10664-012-9208-x.

Galindo, J. A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A. M. & Ruiz-Cortés, A. (2019). Automated analysis of feature models: Quo vadis? *Computing*, 101(5), 387–433. doi: 10.1007/s00607-018-0646-1.

Galster, M., Weyns, D., Tofan, D., Michalik, B. & Avgeriou, P. (2014). Variability in Software Systems - A Systematic Literature Review. *IEEE Trans. Software Eng.*, 40(3), 282-306.

Gonçales, L. J., Farias, K. & da Silva, B. C. (2021). Measuring the cognitive load of software developers: An extended Systematic Mapping Study. *Inf. Softw. Technol.*, 136, 106563. doi: 10.1016/j.infsof.2021.106563.

Guéhéneuc, Y.-G. (2006). TAUPE: towards understanding program comprehension. *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, pp. 1–es.

Heradio, R., Perez-Morago, H., Fernández-Amorós, D., Cabrerizo, F. J. & Herrera-Viedma, E. (2016). A bibliometric analysis of 20 years of research on software product lines. *Information & Software Technology*, 72, 1–15. doi: 10.1016/j.infsof.2015.11.004.

Holmqvist, K. & Andersson, R. (2017). *Eye tracking: a comprehensive guide to methods, paradigms, and measures*. CreateSpace.

Holmqvist, K., Nyström, M. & Mulvey, F. (2012). Eye tracker data quality: What it is and how to measure it. *Eye Tracking Research and Applications Symposium (ETRA)*. doi: 10.1145/2168556.2168563.

Jacob, R. J. & Karn, K. S. (2003). Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. In *The mind's eye* (pp. 573–605). Elsevier.

Jeanmart, S., Gueheneuc, Y.-G., Sahraoui, H. & Habra, N. (2009). Impact of the visitor pattern on program comprehension and maintenance. *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 69–78. doi: 10.1109/E-SEM.2009.5316015.

Kang, K., Cohen, S., Hess, J., Novak, W. & Peterson, A. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (Report n°CMU/SEI-90-TR-21).

Karn, K. S., Ellis, S. & Juliano, C. (1999). The hunt for usability: tracking eye movements. *CHI'99 extended abstracts on Human factors in computing systems*, pp. 173–173.

Lazar, J., Feng, J. & Hochheiser, H. (2017). *Research Methods in Human-Computer Interaction, 2nd Edition*. Morgan Kaufmann. Retrieved from: https://www.sciencedirect.com/science/book/9780128053904.

Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T. & Saake, G. (2017). *Mastering Software Variability with FeatureIDE*. Springer. doi: 10.1007/978-3-319-61443-4.

Melo, J., Narcizo, F. B., Hansen, D. W., Brabrand, C. & Wasowski, A. (2017). Variability through the eyes of the programmer. *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, pp. 34–44. doi: 10.1109/ICPC.2017.34.

Obaidellah, U., Haek, M. A. & Cheng, P. C. (2018). A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Comput. Surv.*, 51(1), 5:1–5:58. doi: 10.1145/3145904.

Peterson, C. S., Saddler, J. A., Blascheck, T. & Sharif, B. (2019). Visually Analyzing Students' Gaze on C++ Code Snippets. *2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)*, pp. 18–25. doi: 10.1109/EMIP.2019.00011.

Petrusel, R. & Mendling, J. (2013). Eye-tracking the factors of process model comprehension tasks. *International Conference on Advanced Information Systems Engineering*, pp. 224–239.

Pohl, K., Bockle, G. & van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer.

Santos, A. R., do Carmo Machado, I., de Almeida, E. S., Siegmund, J. & Apel, S. (2019). Comparing the influence of using feature-oriented programming and conditional compilation on comprehending feature-oriented software. *Empir. Softw. Eng.*, 24(3), 1226–1258. doi: 10.1007/s10664-018-9658-x.

Schulze, S., Liebig, J., Siegmund, J. & Apel, S. (2013). Does the discipline of preprocessor annotations matter?: a controlled experiment. *Generative Programming: Concepts and Experiences, GPCE'13, Indianapolis, IN, USA - October 27 - 28, 2013*, pp. 65–74. doi: 10.1145/2517208.2517215.

Seel, N. M. (2017). Model-based learning: A synthesis of theory and research. *Educational Technology Research and Development*, 65(4), 931–966.

Sharafi, Z., Marchetto, A., Susi, A., Antoniol, G. & Guéhéneuc, Y.-G. (2013). An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 33–42.

Sharafi, Z., Soh, Z. & Guéhéneuc, Y. (2015). A systematic literature review on the usage of eye-tracking in software engineering. *Inf. Softw. Technol.*, 67, 79–107. doi: 10.1016/j.infsof.2015.06.008.

Sharafi, Z., Sharif, B., Guéhéneuc, Y., Begel, A., Bednarik, R. & Crosby, M. E. (2020). A practical guide on conducting eye tracking studies in software engineering. *Empir. Softw. Eng.*, 25(5), 3128–3174. doi: 10.1007/s10664-020-09829-4.

Sharif, B. & Maletic, J. I. (2010). An eye tracking study on the effects of layout in understanding the role of design patterns. *2010 IEEE International Conference on Software Maintenance*, pp. 1–10.

Siegmund, J. (2012). *Framework for measuring program comprehension*. (Ph.D. thesis, Magdeburg, Universität, Diss., 2012).

Soh, Z., Sharafi, Z., Van den Plas, B., Porras, G. C., Guéhéneuc, Y.-G. & Antoniol, G. (2012). Professional status and expertise for UML class diagram comprehension: An empirical study. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pp. 163–172.

Tabachnick, B. G. & Fidell, L. S. (2013). *Using Multivariate Statistics (6th ed.)*. Boston, MA: Pearson.

Thüm, T., Apel, S., Kästner, C., Schaefer, I. & Saake, G. (2014). A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.*, 47(1), 6:1–6:45. doi: 10.1145/2580950.

Uddin, M. S., Gaur, V., Gutwin, C. & Roy, C. K. (2015). On the comprehension of code clone visualizations: A controlled study using eye tracking. *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 161–170.

Valtakari, N. V., Hooge, I. T. C., Viktorsson, C., Nyström, P., Falck-Ytter, T. & Hessels, R. S. (2021). Eye tracking in human interaction: Possibilities and limitations. *Behav Res Methods*, 53(4), 1592–1608. doi: 10.3758/s13428-020-01517-x.

Vyas, G. & Sharma, A. (2016). Empirical Evaluation of Metrics to Assess Software Product Line Feature Model Usability. *International Journal of Science, Engineering and Computer Technology*, 6(2), 82.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C. & Regnell, B. (2012). *Experimentation in Software Engineering*. Springer. doi: 10.1007/978-3-642-29044-2.

Yusuf, S., Kagdi, H. & Maletic, J. I. (2007). Assessing the Comprehension of UML Class Diagrams via Eye Tracking. *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pp. 113–122. doi: 10.1109/ICPC.2007.10.