# Understanding the Developer and User Perspectives of Design Pattern Detection Tools

Rodrigo Moreira[1†], Eduardo Fernandes[1†], Eduardo Figueiredo[1], Filipe Fernandes[2*]

[1*]Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil.
[2]Department of Computer Science, Federal Institute of Southeast of Minas Gerais, Manhuaçu, Brazil.

*Corresponding author(s). E-mail(s): filipe.fernandes@ifsudestemg.edu.br;
Contributing authors: rodrigo.moreira@dcc.ufmg.br; eduardomorfernandes@gmail.com; figueiredo@dcc.ufmg.br;
†These authors contributed equally to this work.

**Abstract**

**Context**: Design Pattern Detection (DPD) tools are useful to support to the comprehension and maintenance of software systems. Although several DPD tools have been introduced over the years, they typically focus on a limited set of design patterns and programming languages. **Objective**: This paper aims to investigate (i) the reasons that motivate DPD tool designers to target specific design patterns and programming languages, and (ii) how potential users perceive the usefulness of DPD tools in practical software development scenarios. **Method**: We conducted two online surveys. For the first survey, we reached out to designers of 42 DPD tools selected from a systematic literature review to obtain their perspectives on design decisions about pattern and language coverage, receiving 22% of such responses. For the second survey, we recruited 28 student and senior developers to help us understand their expectations, perceived benefits, and concerns related to the use of DPD tools. **Results**: Within our sample, the findings suggest that participating tool designers often prioritize design patterns whose internal structure facilitates automated detection, while language support is frequently motivated by popularity and expected demand. From the perspective of tool users, DPD tools are expected to support development activities, such as program comprehension and software quality improvement. Unfortunately, usability difficulties, limited accuracy, and insufficient documentation often discourage them

from adopting a tool. **Conclusions**: The responses suggest that some design decisions reported by participating DPD tool designers are aligned with practical and industrial considerations. However, the recurring lack of adequate documentation and usability support is a major barrier to wider usage. Taken as indicative evidence, these results suggest that future DPD tools should better balance detection capabilities and usability concerns, especially to meet the need of less experienced developers. Given the limited number of tool-designer respondents, conclusions about design rationale should be interpreted as indicative rather than representative of all DPD tool designers.

**Keywords:** software design patterns, automated detection, survey, qualitative analysis

# 1 Introduction

Design patterns are well-documented and reusable solutions for recurring problems of software design (Gamma et al, 1995). Hundreds of design patterns have been implemented by software developers while structuring their software systems (Zanoni et al, 2015). Composite, Facade, and Singleton are examples of traditional design patterns compiled by the Gang of Four (GoF)'s book (Gamma et al, 1995). The implementation of design patterns is expected to prevent the software design degradation, which often affects systems at the source code and architecture levels (Ahmed et al, 2015; Ahmad et al, 2023; Garcia et al, 2005). Therefore, constantly detecting and monitoring the implementation of design patterns may be critical for the long-term success of a software project (Kim, 2025).

Design Pattern Detection (DPD) has many applications in software industry, e.g., source code analysis (Kim et al, 2016) and design decision making (Zanoni et al, 2015). However, manually performing DPD may be quite challenging due to the large size and high complexity of modern systems (Russo, 2024; Wang and Tzerpos, 2005). Several DPD tools have been proposed to automate the detection of different design patterns, thereby making the detection faster and less error prone. Our previous systematic literature review (Moreira et al, 2022) provides an extensive catalog of 42 DPD tools published in academic papers over the last two decades. MARPLE-DPD (Zanoni et al, 2015) and PTIDEJ (Moha and Guéhéneuc, 2007) are examples of prestigious DPD tools that detect different design patterns, such as Chain of Responsibility, Composite, Decorator, Prototype, Singleton, and Visitor. More recently, artificial intelligence models, such as Large Language Models (LLM), have been used to verify the implementation of design patterns (Kim, 2025; Nguyen-Duc et al, 2025).

In fact, our literature review (Moreira et al, 2022) revealed that DPD tool designers target very specific design patterns and programming languages. For instance, 67% of the tools detect Composite, while only 19% detect Facade. Additionally, 69% of the tools are compatible with Java systems, while no tool is compatible with trending programming languages, e.g., JavaScript and Python. Unfortunately, we do not know why DPD tool designers have targeted specific design patterns and programming languages. More critically, we lack evidence on the expected usefulness of DPD tools from

the perspective of potential tool users. It could be that the 42 DPD tools we found fall short in meeting users' expectations.

In this paper, we introduce a survey-based study with the purpose of addressing the aforementioned knowledge gaps. We relied on strict guidelines (Punter et al, 2003; Molléri et al, 2016) to conduct two online surveys aimed at capturing the perceptions of tool designers and tool users about DPD tools. For the first survey, we emailed all authors of the 42 DPD tools found in our literature review (Moreira et al, 2022). We asked authors to answer questions regarding the reasons for targeting specific design patterns and programming languages with their tools. Designers of nine out of the 42 DPD tools (i.e., 21.4%) responded to our survey, providing a limited but informative set of responses for tools proposed over two decades.For the second survey, we recruited 28 participants, including 17 Computer Science students and 11 senior developers, who are potential users of DPD tools. We asked them about the expected usefulness of DPD tools.

Inspired by Grounded Truth procedures (Stol et al, 2016), we manually extract topics from the open questions answered by the survey participants. Additionally, we followed strict guidelines (Cruzes and Dyba, 2011; Molléri et al, 2016) to generate coding-based categorizations from the topics aimed at organizing the responses about reasons to target specific contexts in DPD tool design as well as the expected usefulness of these tools. We summarize below our main study findings and their implications.

- We found five different reasons for targeting specific design patterns in the proposal of DPD tools. Among the participating tool designers, most reported that internal aspects of a design pattern (e.g., the way it is structured in the source code) favored its detection. Additionally, popularity and abstraction level are recurring reasons in our sample for decisions about detecting design patterns in specific programming languages.
- Some participating tool designers showed interest in proposing tools for a wider scope of design patterns and programming languages. However, this would depend on the availability of enabling technologies (e.g., friendly compilers) in addition to the practitioner's demands. This finding could inspire engineers in filling gaps in terms of these technologies.
- Participating tool designers reported awareness of their DPD tools being used for program analysis tasks, especially program comprehension. This finding meets the expected benefits of using DPD tools reported by the potential tool users. According to the tool users, other benefits include software quality improvement (e.g., via code organization and cleaning).
- Potential tool users claim that tool limitations (e.g., low accuracy and lack of documentation), as well as the inherent difficulty to use the tool, are key barriers to the DPD tool adoption. This finding is aligned with the barriers of adopting other types of automated tools discussed in previous work (Alkharabsheh et al, 2019; Johnson et al, 2013).

The remainder of this paper is organized as follows. Section 2 describes our study design. Sections 3 and 4 present the results of our surveys with DPD tool designers and potential tool users, respectively. Section 5 summarizes the main implications of

our findings. Section 6 discusses the findings in light of the broader context of DPD research and practice. Section 7 discusses threats to the validity of the study. Section 8 compares our study with previous work. Finally, Section 9 concludes the paper and outlines future work.

# 2 Study design

This section provides details regarding our study design. Section 2.1 introduces the study goal and research questions (RQs). Section 2.2 presents the structure of our two surveys: the survey with tool designers and the survey with potential tool users. Section 2.3 describes our study participants for each survey. Lastly, Section 2.4 describes our data analysis procedures.

## 2.1 Goal and research questions

We relied on the Goal-Question-Metric template (Basili and Rombach, 1988) to define our study goal as follows: the study aims to *analyze* the perceptions of tool designers and potential tool users about DPD tools; *for the purpose of* understanding the extent in which existing DPD tools meet the needs of tool users, as well as assisting tool designers in proposing novel tools; *with respect to* i) what reasons have led tool designers to target specific design patterns and programming languages and ii) the expected usefulness of DPD tools from the perspective of potential tool users; *from the point of view of* tool designers who have contributed to academic publications and student or senior developers who are potential tool users; *in the context of* 42 tools summarized in our recent systematic literature review (Moreira et al, 2022). We consider student developers those who are still enrolled in their Computer Science degree programs and whose experience is limited to small projects developed in academic contexts. Senior developers, on the other hand, are professionals who already work in the software development industry.

We defined the two RQs below to drive our research.

**RQ$_1$:** *What is the design rationale behind the proposal of the existing DPD tools?* – Our literature review (Moreira et al, 2022) catalogs 42 DPD tools published over the last two decades. We found a certain bias in the design patterns these tools detect: at least 64% of the tools detect Composite or Observer, while only a few tools detect other relevant patterns, such as Mediator (31%) and Facade (19%). We also found biased the choice for target languages: 69% of the tools target Java, while we found no tool targeting trending languages, such as JavaScript and Python. The lack of documentation regarding the reasons why tool designers have targeted specific patterns and languages motivates our RQ. We believe that RQ$_1$ can provide initial evidence on types of technical support that may help tool designers target other patterns and languages of relevance in industry.

**RQ$_2$:** *How useful do potential tool users expect DPD tools to be?* – A few previous studies (Forward and Lethbridge, 2002; Johnson et al, 2013; Liebel et al, 2017; Witschey et al, 2015) rely on surveys or interviews to investigate the expected usefulness of different software development tools. These studies highlight the importance of understanding an eventual mismatch between the tool designers' effort and the

practical needs of developers. To the best of our knowledge, there is no survey-based empirical study aimed at understanding the expected usefulness of DPD tools. We advocate that acquiring such understanding is essential to propose novel DPD tools (or refine existing tools) that are able to meet the current needs of potential users. With $RQ_2$, we capture contexts in which student or senior developers expect DPD tools to be useful, as well as benefits and barriers to the use of these tools.

## 2.2 Survey structure

Surveys are aimed at collecting information from people as a means to characterize their opinion, knowledge, or behavior (Wohlin et al, 2012). Surveys help researchers derive conclusions from a sample of participants that does not necessarily reflect an entire population. Thus, survey replication is highly recommended. We opted for a survey-based study due to our positive previous experiences (Nascimento et al, 2021; Oliveira et al, 2020; Santos et al, 2024). We relied on strict guidelines for conducting online surveys (Punter et al, 2003; Molléri et al, 2016) to propose two complementary surveys: one targeting tool designers and one targeting potential tool users. We opted for short surveys to prevent candidate participants from being exhausted in answering our survey. We describe below the structure of each survey.

**Survey with tool designers:** Table 1 presents the five open questions of the survey with tool designers (D1 to D5). We opted not to ask demographic information directly to the tool designers as this information was publicly available on their websites. We customized the survey according to the specific information of each tool (see text between brackets). D1 captures the reasons why the tool designers decided to detect specific design patterns. D2 captures the designers' intent to detect other design patterns they believe to be relevant. D3 and D4 are similar to D1 and D2 but target programming languages rather than design patterns. D5 capture the designers' knowledge of the usage of their tools.

**Table 1**: Structure of the survey with tool designers

| ID | Question |
|---|---|
| D1 | Your tool detects [list of design patterns]. Why did you choose them? |
| D2 | Would you implement detection tools for other design patterns? If so, which design patterns would you support? Why? |
| D3 | Your tool detects design patterns in systems implemented in [list of programming languages]. Why did you choose them? |
| D4 | Would you implement detection tools for other languages? If so, what languages would you support? Why? |
| D5 | In which contexts of software development has your tool been used (e.g. adding a new feature to the system, enhancing an existing feature or code, removing an obsolete feature, fixing a bug, etc.)? |

**Survey with tool users:** Table 2 lists the eight questions of our survey with tool users: U1 to U5 are closed questions, while U6 to U8 are open questions. U1 to U4 capture demographic information aimed at helping us to understand the possible generalization of our findings. These questions cover the programming language most

used by the user (U1), years of experience as a programmer (U2), their current main role as a developer (U3), and familiarity with design patterns (U4). U5 to U8 capture the expected usefulness of DPD tools: contexts in which tool users would consider using DPD tools (U5), expected benefits of using a DPD tool (U6), reasons for not using a DPD tool (U7), and design patterns that are worth detecting via tool (U8).

**Table 2**: Structure of the survey with potential tool users

| ID | Question | Possible answers |
|---|---|---|
| U1 | Which programming language do you use mostly on a daily basis? | JavaScript; Python; Java; C#; PHP; other |
| U2 | How many years of experience do you have as a programmer? | Up to 3 years; 3 - 5 years; 5 - 10 years; more than 10 years |
| U3 | What is your current main role? | Back-end developer; front-end developer; full stack developer; tester; other |
| U4 | How familiar are you with design patterns? | Not familiar; somewhat familiar; fairly familiar; very familiar |
| U5 | In which contexts of software development would you consider using a design pattern detection tool? | Adding a new feature to the system; enhancing an existing feature or code; removing an obsolete feature; fixing a bug; other |
| U6 | What benefits do you expect from using a design pattern detection tool? | N/A (open question) |
| U7 | What would prevent you from using a design pattern detection tool? | N/A (open question) |
| U8 | What design patterns do you believe are worth detecting through a tool? Why? | N/A (open question) |

## 2.3 Participant characterization

As discussed in Section 2.2, this paper introduces an empirical study based on two online surveys, each targeting a specific group of participants. We describe below our participant recruitment procedures and characterize each group.

**Group A:** *Tool designers* – Our literature review (Moreira et al, 2022) summarizes 42 DPD tools. We used the papers that introduced each tool to extract contact information of the respective tool designers. We extracted the names and email addresses of all authors credited in each paper. The 42 tools were published over a span of 20 years, so that certain email addresses were likely invalid. This time span is part of the systematic catalog investigated in the study rather than a sampling flaw: the catalog intentionally covers the historical development of DPD tools. We prevented this issue by searching for the latest papers published by each author on Google Scholar[1]. We then extracted the email addresses provided in these papers; we also extracted the email addresses informed in the author's website whenever its link was provided by the author's profile on Scholar.

We emailed our survey to all authors (i.e., tool designers). We tried all email addresses extracted per author and, if all addresses were invalid and our messages returned, we simply discarded the author from Group A. We extracted 95 email

[1] https://scholar.google.com

addresses from a total of 108 tool designers. We waited two months for responses to the survey, so that tool designers (especially designers of older tools) had enough time to gather information about the design decisions of their previous work. Tool designers of nine out of the 42 tools responded to our survey, one designer by tool, thereby covering 21.4% of the tools. Given the exploratory purpose of the survey and the small number of responding tools, we did not perform an age-stratified comparison of responses by publication year; such a comparison could overstate differences between older and newer tools.

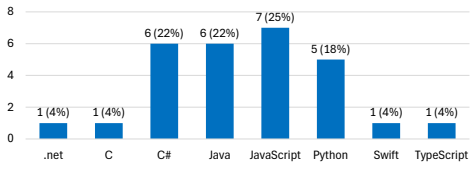
**Demographic information for Group A:** Eight out of the nine survey participants (89%) hold a PhD in Computer Science. Three participants are working in the industry and six are professors at different universities. Regarding their positions in the coauthors' list of their respective papers, four of them are first authors, four are second authors and one is third author. Thus, although the sample is small, the survey participants are qualified to provide information about the decisions behind the proposal of their DPD tools.

**Group B:** *Tool users* – We had no reference list of software developers who are potential users of DPD tools and, so, the universe of possible survey participants is very broad. Aimed at achieving a diverse group of participants in terms of background, we opted for recruiting potential tool users based on opportunistic broadcasting. We sent the survey link to three groups of software developers on WhatsApp[2], which is an instant messaging platform largely used in Brazil. These groups were Brazilian developer groups reachable through the authors' contact networks. However, we did not systematically record whether each group was professional, corporate, academic, amateur, or mixed. We encouraged all developers to forward our survey link to other developers and similar WhatsApp groups. Because we could not track all developers reached by our survey link, it is impossible to compute response rate. Still, we obtained responses from four senior developers in this initial recruitment round. To increase the number of senior developers in our sample, we conducted an additional recruitment round specifically targeting senior developers. In this round, we sent the survey to previous members of our software engineering research group, including previous doctoral and postdoctoral researchers who matched the profile with large software development experience. This additional round resulted in seven further responses, leading to 11 senior developers in total. We also forwarded our survey link to undergraduate Computer Science students enrolled in the Software Engineering course at the Federal University of Minas Gerais (UFMG) – the affiliation of two authors of this paper. We gave each student a month to respond the survey, enough time for an opinion-based survey. A total of 17 students responded to our survey, resulting in 28 participants in Group B.

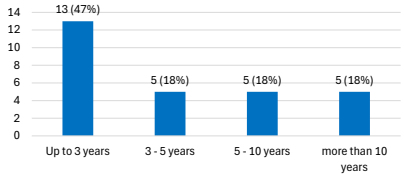
**Demographic information for Group B:** Figure 1 summarizes demographic information of the potential tool users collected from survey questions U1 to U4 (Table 2). Figure 1a shows that the programming languages more often used by the potential tool users on a daily basis are JavaScript (25% of the users), C# (21%), Java (21%), and Python (18%). The remaining languages, .net, C, Swift, and TypeScript, were each reported by one participant (4%). Figure 1b indicates that 13 participants

[2] https://www.whatsapp.com/

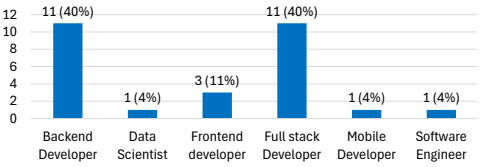
(47%) have up to three years of programming experience, while the remaining participants are evenly distributed across the ranges of three to five years, five to ten years, and more than ten years, with five participants (18%) in each range. Figure 1c shows that backend developers and full stack developers are the most common roles, each reported by 11 participants (40%). Frontend developers account for three participants (11%), while data scientist, mobile developer, and software engineer were each reported by one participant (4%). Finally, Figure 1d depicts that 23 participants (82%) reported at least some familiarity with design patterns, including 11 somewhat familiar participants (39%), eight fairly familiar participants (29%), and four very familiar participants (14%). The remaining five participants (18%) reported not being familiar with design patterns.



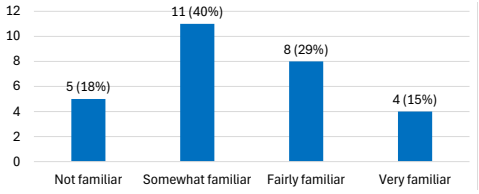

(a) Most used programming language




(b) Years of experience as a programmer




(c) Current main role




(d) Familiarity with design patterns

**Fig. 1**: Demographic information of participants of the survey with tool users

## 2.4 Data analysis procedures

Figure 2 depicts our data analysis procedures. We defined a total of seven steps, which range from quantitative to qualitative analysis of the survey data. We relied on strict guidelines for conducting empirical research in software engineering (Cruzes and Dyba, 2011; Punter et al, 2003; Stol et al, 2016; Wohlin et al, 2012) to support the definition of our procedures.

Only a few of our survey questions (U1 to U5) are closed questions, whose data are typically more straightforward to analyze. The first author of this paper computed frequency of survey answers (e.g., programming languages reported on survey question U1) for the descriptive analysis in Step 1.1. The same author plotted the bar charts summarized in Figure 1 in Step 1.2. All authors of this paper contributed to performing double checking in Step 1.3.

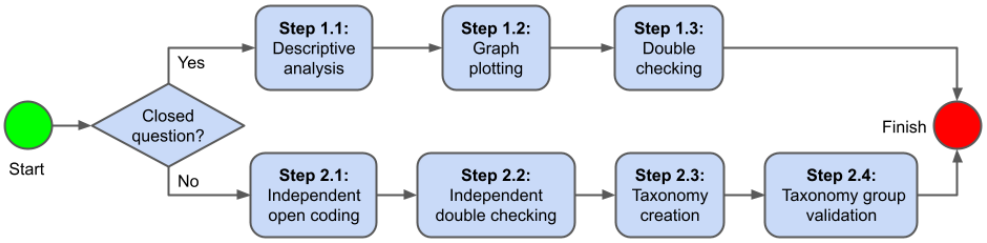



**Fig. 2**: Data analysis procedures for closed survey questions

The remaining survey questions are open questions and we opted for manual analysis to achieve accuracy in the data analysis. For such analysis, one author of this paper performed Step 2.1 to extract textual topics (i.e., codes) from each open question response via open coding (Stol et al, 2016). A different author double checked the codes and suggested fixes in Step 2.2, thereby mitigating subjective biases. All authors discussed the cases of disagreement to reach consensus. Thus, our coding procedure was consensus-based rather than based on independent parallel coding with quantitative agreement statistics. Step 2.3 also involved two authors of this paper working together to create all coding-based categorizations based on the codes extracted for each survey question. We followed an iterative process as suggested by previous work (Cruzes and Dyba, 2011). All authors of this paper also helped validate the coding-based categorizations created for each survey question in Step 2.4.

We provide additional comments regarding open coding (Step 2.1) and coding-based categorization construction (Step 2.3) as follows. Not all survey responses provided a valid code during the open coding; some survey responses provided more than one valid code. Thus, the number of codes grouped by coding-based categorization is not necessarily equal to the number of survey participants. We decided to shorten the coding-based categorizations presented in this paper for the sake of simplicity. Our study replication package, whose URL is provided in the Data Availability statement at the end of the paper, includes the full coding-based categorizations.

Following previous work in empirical software engineering (Olsen and Fox, 2019; Ralph, 2019), we use the term coding-based categorization to describe the structures derived from open-ended survey responses. Our coding procedure organized participants' answers into general and concrete categories, but we do not claim theoretical saturation or that these categories constitute saturated taxonomies or theory-level contributions. Some concrete categories have low frequencies, such as only one or two mentions. Therefore, these categories should be interpreted as isolated or low-frequency mentions that help characterize the range of participants' perceptions, rather than stable or generalizable classes of phenomena. This terminology is consistent with the exploratory nature of our surveys and with the use of questionnaire data as a single source of qualitative evidence.

# 3 Results of the survey with tool designers

This section reports the results of the survey with tool designers as follows. Section 3.1 presents the reasons why participating tool designers targeted specific design patterns. Section 3.2 discusses the reasons why participating tool designers targeted specific programming languages. Section 3.3 presents the contexts in which participating tool designers know that their tools have been used.

## 3.1 Reasons to detect specific design patterns

**Survey question D1:** Figure 3 summarizes the reasons reported by participating tool designers to target specific design patterns. We found five concrete reasons (blue rectangles with continuous borders), which we grouped into two general reasons (white rectangles with dashed borders): **External aspects of design patterns** and **Internal aspects of design patterns**. We annotated each reason with its frequency; such frequency corresponds to the number of participants (out of the nine tool designers) who mentioned a reason.

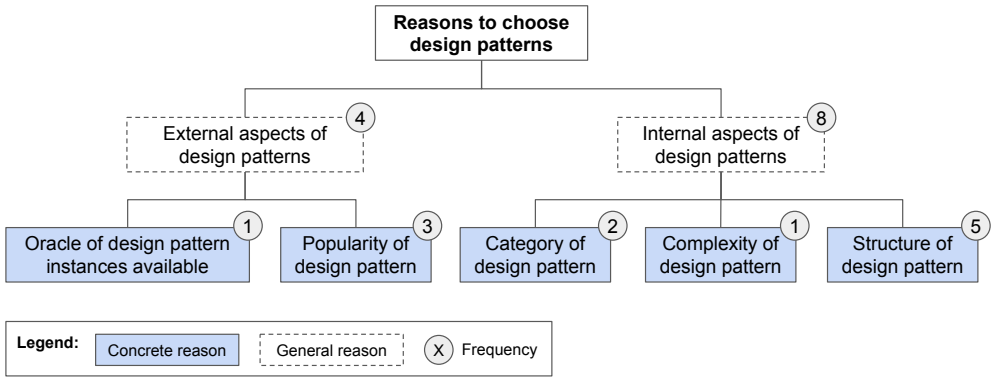



**Fig. 3**: Coding-based categorization of reasons to target specific design patterns

Table 3 describes the reason founds and illustrates each reason with quotes from the survey responses. Four responses refer to **External aspects of design patterns**, whose reasons include: **Oracle of design pattern instances available** (R1) and **Popularity of design pattern** (R2). The latter is the the most frequent one. The multiple mentions to R2 suggest that some participating tool designers may have tried to align their work with current industry trends. On the other hand, eight responses refer to **Internal aspects of design patterns**, whose reasons include: **Category of design pattern** (R3), **Complexity of design pattern** (R4), and **Structure of design pattern** (R5); this one is the most frequent. Based on the mentions to R5 within our sample, we infer that participating tool designers often prioritized the detection of design patterns whose internal structure is easier to characterize.

**Survey question D2:** We asked tool designers whether they would implement tools for detecting design patterns beyond the current scope of their DPD tools. Out

**Table 3**: Details of reasons to target specific design patterns

| ID | Description | Sample quote |
|---|---|---|
| R1 | Reference list of (typically validated) instances of design pattern instances detected on a software system | *"we selected patterns that were documented in the selected subject systems"* – Tool designer TD6 |
| R2 | Frequency in which a design pattern is discussed (by either researchers or practitioners) or implemented in industry | *"GoF patterns are well known and widely used in software design"* – Tool designer TD4 |
| R3 | Category assigned to a design pattern (e.g., based on its purpose of internal structure) | *"Because [the design patterns] are representative of the main categories/-families of design patterns"* – Tool designer TD1 |
| R4 | Inherent difficulty in understanding the concept or implementation of a design pattern | *"[The design patterns] are of varying complexities"* – Tool designer TD1 |
| R5 | Internal composition of a design pattern implemented in the source code of a system | *"because [these design patterns] are very similar with regard to the structural properties (classes, relations between them)"* – Tool designer TD5 |

of the nine tool designers surveyed, four responded "Yes", two responded "No", and three did not respond this optional question. Examples of design patterns mentioned by the tool designers who responded affirmatively include Flyweight, Interpreter, and Iterator. We discuss below the reasons reported by responding tool designers to target these specific design patterns in a hypothetical future implementation.

Figure 4 summarizes the reasons reported by the tool designers on that matter. We found only two different reasons (blue rectangles with continuous borders). We labeled the first reason as **Completeness of the tool** (R1) by inferring that some respondents aimed at completing the detection scope of the tool given a literature reference. R1 can be illustrated by the tool designer TD9 who reported that their *"original goal was to support all 23 [design patterns of the GoF's book]"*. Such interest may be due to the prestige that the mentioned book (Gamma et al, 1995) has in the context of object-oriented programming. We labeled the second reason as **Practitioner's demands** (R2) as suggested by the interest of designer TD5 in implementing tools for detecting *"any pattern that is relevant to a developer"*.

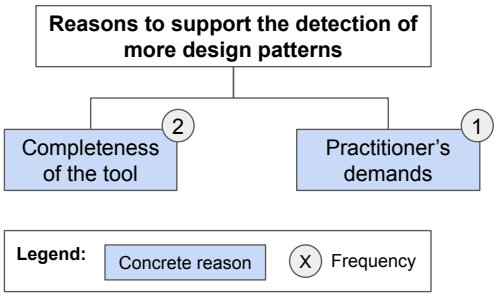



**Fig. 4**: Reasons to support the detection of more design patterns

## 3.2 Reasons to support specific programming languages

**Survey question D3:** Figure 5 depicts the reasons reported by participating tool designers to target specific programming languages. We found five concrete reasons (blue rectangles with continuous borders), which we grouped into two general reasons (white rectangles with dashed borders): **Aspects of the language** and **Other aspects**. We annotated each reason with its frequency, i.e., the number of participants (out of the nine tool designers) who mentioned a reason.

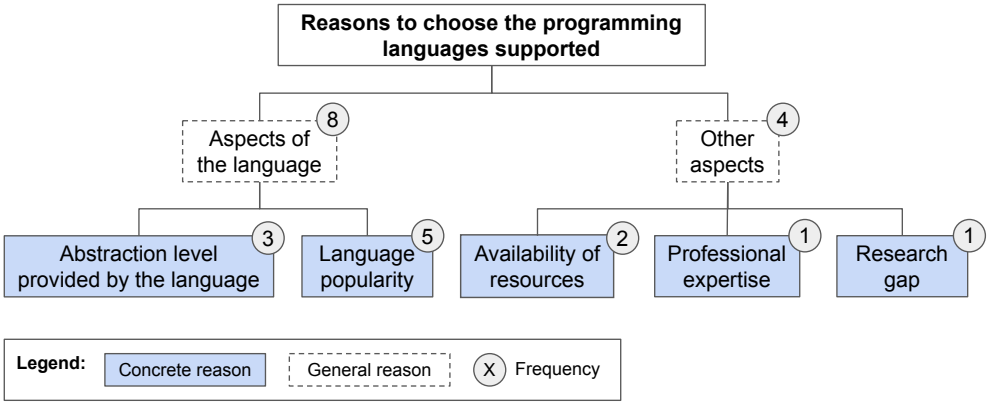



**Fig. 5**: Reasons to target specific programming languages

Table 4 describes the reasons found and illustrates each reason with quotes from the survey responses. Eight participants refer to **Aspects of the language**, whose two reasons are: **Abstraction level provided by the language** (R1) and **Language popularity** (R2). The high frequency of mentions to R2 suggests that some participating tool designers attempted to align their work with industry trends. Still, R1 plays an important role in deciding for performing DPD on a specific language. On the other hand, four participants refer to **Other aspects**, which include the following reasons: **Availability of resources** (R3), **Professional expertise** (R4), and **Research gap** (R5). These reasons suggest that convenience may have influenced some design decisions reported in our sample.

**Survey question D4:** We asked tool designers whether they would implement tools for detecting design patterns on systems implemented in programming languages other than the languages already supported by their DPD tools. Out of the nine tool designers surveyed, seven responded "Yes", one responded "No", and one did not respond the question. Examples of programming languages mentioned by the tool designers who responded affirmatively are C, C++, C#, and Python. These four examples are among the five most popular languages (e.g., in TIOBE Index).

Figure 6 summarizes the reasons reported by participating tool designers to target the aforementioned specific languages in an hypothetical future. We found three concrete reasons in total (blue rectangles with continuous borders). We labeled the first

**Table 4**: Details of reasons to target specific programming languages

| ID | Description | Sample quote |
|---|---|---|
| R1 | Abstraction level provided by a programming language to support the implementation of software features | *"[This language] provided the right level of abstraction"* – Tool designer TD1 |
| R2 | Frequency in which a programming language is adopted by either researchers or software developers | *"[Language X] was the most popular object-oriented programming language at that time"* – Tool designer TD2 |
| R3 | Availability of technologies compatible with or designed for manipulating a programming language | *"because we had created [auxiliary tool name], the tool we used as backend for [our DPD tool]"* – Tool designer TD5 |
| R4 | Practical experience with the use of a programming language | *"Because of my experience with [Language X] use and [Language X] compilers"* – Tool designer TD5 |
| R5 | Research field yet to be addressed by scientific research | *"we were the first model to address this research gap"* – Tool designer TD4 |

reason as **Compiler related concerns** (R1), which refers to the compilation environment available for a specific language. This reason is illustrated by the mention to *"any [...] language that has an open source, reliable, well documented and understandable compiler"* made by tool designer TD5. We labeled the second reason as **Language popularity** (R2), which can be illustrated by the mention of *"any widely used language"* also made by tool designer TD5. The third reason was labeled as **Practitioner's demands** (R3), which is illustrated by a mention of *"whatever language [that] is being used by practitioners"* from the response of tool designer TD1. R2 and R3 are consistent with our interpretation that some participating designers were aware of industry trends.

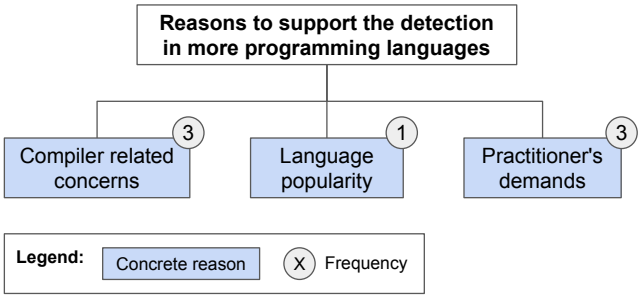



**Fig. 6**: Reasons to support the detection in more programming languages

## 3.3 Reported contexts of tool usage

**Survey question D5**: Figure 7 summarizes the contexts reported by participating tool designers in which their tools have been used. We found five concrete contexts (blue rectangles with continuous borders). Four out of the five contexts were grouped into two general contexts (white rectangles with dashed borders): **Program analysis**

and **Source code changes**. We annotated each context with its frequency; such frequency is the number of participants (out of the nine tool designers) who reported a context. We discuss below each context with sample quotes extracted from the survey responses.

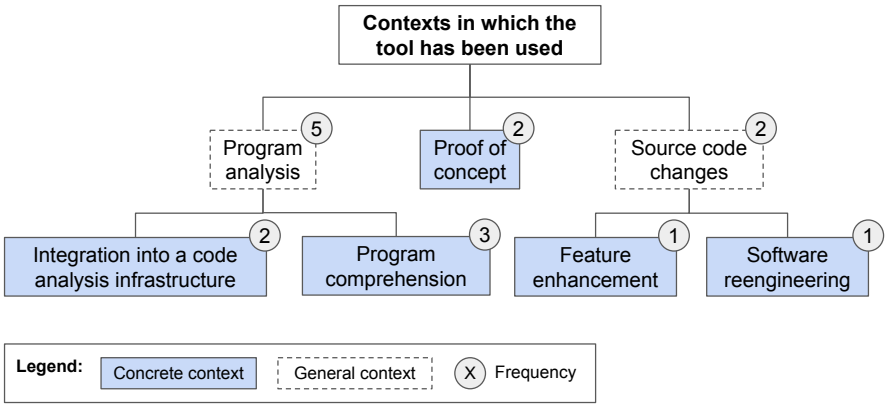



**Fig. 7**: Coding-based categorization of contexts in which tools have been used

Table 5 describes and illustrates each context reported by tool designers. Five responses refer to **Program analysis**, whose contexts were labeled as: **Integration into a code analysis infrastructure** (C1) and **Program comprehension** (C2), the latter being the most frequent context. Two responses refer to **Source code changes**, and contexts include: **Feature enhancement** (C3) and **Software reengineering** (C4). C3 and C4 suggest that DPD tools have been used to assist developers in making design decisions on their systems. Finally, two responses refer to **Proof of concept** (C5).

# 4 Results of the survey with tool users

This section reports the results of our survey with tool users. Section 4.1 presents the contexts in which tool users would use a DPD tool. Section 4.2 discusses expected benefits of using a tool. Section 4.3 discusses barriers that would prevent tool users from using a tool. Lastly, Section 4.4 discusses why tool users believe that detecting certain design pattern is worthwhile.

## 4.1 Potential tool use contexts

**Survey question U5:** Figure 8 summarizes the responses of tool users regarding contexts in which they would consider using a DPD tool. U5 is a closed question and each tool user could report more than one context. Tool users were also allowed to mention additional contexts in the "Other" field, but none of our survey participants did that. Our 28 survey participants provided us with 58 options to this question in total. We discuss below our main observations.

**Table 5**: Details of contexts in which tools have been used

| ID | Description | Sample quote |
|---|---|---|
| C1 | Integration of the DPD tool into a larger code analysis infrastructure | *"[A company] wanted to implement a search engine [...] and the search engine would fetch design pattern instances already implemented in the company's codebase [based on the reports of our DPD tool]"* – Tool designer TD6 |
| C2 | Process of understanding the internal code structure and execution of a software system | *"[Our tool] is used to provide concrete information for developers and technical managers about the usage of design patterns in the implemented code"* – Tool designer TD4 |
| C3 | Structural or functional improvement of existing software features | *"[Our tool has been used for] enhancing an existing feature or code"* – Tool designer TD3 |
| C4 | Reconstruction of an entire software system aimed at improved quality | *"The tool has been used to re-engineer systems and applications"* – Tool designer TD2 |
| C5 | Prototype designed to demonstrate the feasibility of a concept | *"We wanted to demonstrate the capability [of our DPD tool to detect design patterns]"* – Tool designer TD1 |

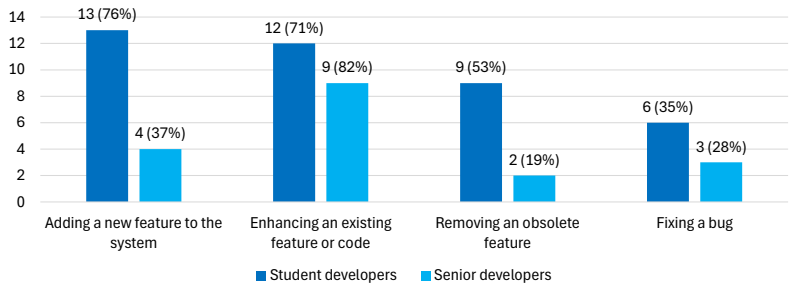



**Fig. 8**: Contexts in which developers would consider using a DPD tool

Regarding responses of students, more than 70% of the 17 participants reported that DPD tools may be useful while either adding new software features (13 responses, 76%) or enhancing existing features (12 responses, 71%). Additionally, nine students (53%) mentioned that DPD tools could help remove obsolete features, which is aligned to the daily software development practices. Curiously, six students (35%) mentioned bug fixing as a context in which using a DPD tool could be useful. These results are expected if we consider the common wisdom that well-structured code can facilitate feature addition or enhancement (Fernandes et al, 2020; Paixao et al, 2019; Santana et al, 2024) and reduce bug-proneness (Palomba et al, 2016). Among the 11 senior developers, feature enhancement was the most frequent context (nine responses, 82%), followed by adding new software features (four responses, 37%), fixing bugs (three responses, 28%), and removing obsolete features (two responses, 19%).

## 4.2 Expected benefits of using a DPD tool

**Survey question U6:** Figure 9 summarizes the benefits expected by the tool users from using a DPD tool. We found a total of ten concrete benefits (blue rectangles with continuous borders), which are grouped into three general benefits (white rectangles with dashed borders): **Code structure**, **Program functioning**, and **Software development**. We annotated the benefits with two different frequencies: triangles represent the frequency of senior developers who reported a specific benefit, while diamonds represent the frequency of students who reported that benefit. We discuss below our major findings followed by quotes from the survey responses.

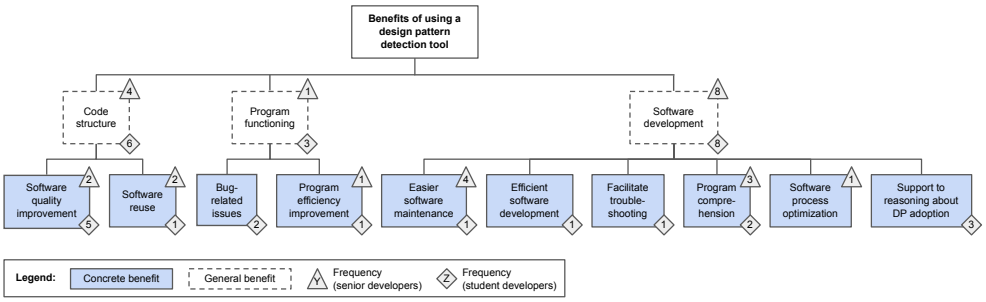



**Fig. 9**: Expected benefits of using a DPD tool

Table 6 describes and exemplifies with quotes each expected benefit. The coding yielded four senior developer and six student developer mentions of **Code structure**, which is related to improving the internal source code structure. The concrete benefits in this category are: **Software quality improvement** (B1), mentioned by two senior developers and five students, and **Software reuse** (B2), mentioned by two senior developers and one student. Additionally, **Program functioning**, a category of benefits associated with program functioning improvements, received one senior developer and three student developer mentions. The concrete benefits include: **Bug-related issues** (B3), mentioned by two students, and **Program efficiency improvement** (B4), mentioned by one senior developer and one student. Finally, eight senior developer and eight student developer mentions were related to **Software development**: **Easier software maintenance** (B5), mentioned by four senior developers and one student; **Efficient software development** (B6), mentioned by one student; **Facilitate trouble-shooting** (B7), mentioned by one student; **Program comprehension** (B8), mentioned by three senior developers and two students; **Software process optimization** (B9), mentioned by one senior developer; and **Support to reasoning about DP adoption** (B10), mentioned by three students.

## 4.3 Barriers to the use of DPD tools

**Survey question U7:** Figure 10 summarizes the barriers mentioned by potential tool users as possible barriers to the adoption of DPD tools. We found a total of ten concrete barriers (blue rectangles with continuous borders), which we grouped

**Table 6**: Details of the expected benefits of using a DPD tool

| ID | Description | Sample quote |
|---|---|---|
| B1 | Internal quality improvement, e.g., via source code cleaning | *"[DPD tools may help] making the code base cleaner for new features"* – Student ST11 |
| B2 | Knowledge or source code reuse | *"[Using a DPD tool may help the] development of similar applications"* – Senior developer SD4 |
| B3 | Dealing with software bugs, e.g., via prevention or fixing | *"[Using a DPD tool may help] prevent bugs"* – Student ST13 |
| B4 | Improved efficiency in terms of program execution time | *"[Using a DPD tool may lead to] increased code velocity"* – Student ST15 |
| B5 | Facilitation of software maintenance tasks | *"A benefit would be an] easy-to-maintain source code that other people can use"* – Student ST3 |
| B6 | Improved efficiency in terms of time spent to develop software | *"[Using a DPD tool] saves time for the developer"* – Student ST4 |
| B7 | Facilitation of the monitoring and fixing of failures, faults or defects | *"[Using a DPD tool allows for] better and more precise troubleshooting"* – Student ST6 |
| B8 | Process of understanding the internal structure and execution of a system | *"[Using a DPD tool] saves time for code understanding"* – Student ST1 |
| B9 | Adjust and improvement of the software development processes | *"[Using a DPD tool may assist] process optimization"* – Senior developer SD2 |
| B10 | Process of reasoning about benefits, drawbacks, and challenges in adopting design patterns | *"[Using a DPD tool may help to] identify [...] which design pattern to use or if the current one used is the most appropriate"* – Student ST4 |

into three general barriers: **Development limitations**, **Tool limitations**, and **Tool usage limitations**. We annotated the barriers with two different frequencies: triangles represent the frequency of senior developers who reported a specific barrier, while diamonds represent the frequency of students who reported that barrier. We discuss below our main findings.

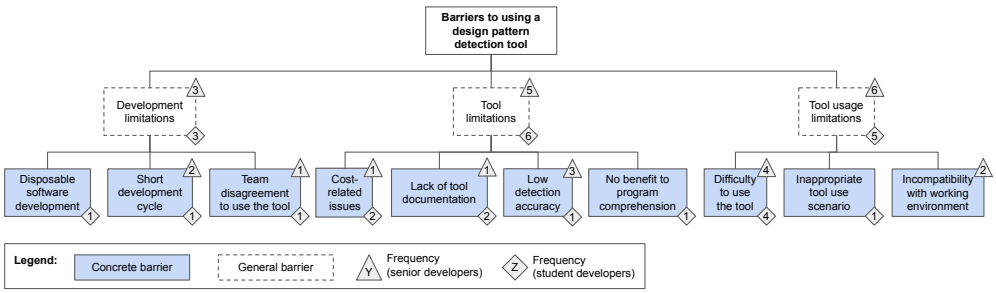



**Fig. 10**: Coding-based categorization of barriers to the use of DPD tools

Table 7 provides descriptions and sample quotes for each barrier. **Development limitations**, which refers to barriers faced over the life cycle of a software system,

received three senior developer and three student developer mentions. The respective concrete barriers are: **Disposable software development** (B1), mentioned by one student; **Short development cycle** (B2), mentioned by two senior developers and one student; and **Team disagreement to tool use** (B3), mentioned by one senior developer and one student. **Tool limitations** received five senior developer and six student developer mentions as a general barrier associated with either technical or functional limitations of the DPD tool. The respective concrete barriers include: **Cost-related issues** (B4), mentioned by one senior developer and two students; **Lack of tool documentation** (B5), mentioned by one senior developer and two students; **Low detection accuracy** (B6), mentioned by three senior developers and one student; and **No benefit to program comprehension** (B7), mentioned by one student. The occurrences of B5 and B6 are red flags because, as our past work suggests (Moreira et al, 2022), i) existing DPD tools are poorly documented and ii) their accuracy is often low.

**Table 7**: Details of barriers to the use of DPD tools

| ID | Description | Sample quote |
|---|---|---|
| B1 | Development of disposable software, e.g., proof of concept or prototype | *"[I would not use a DPD tool in] a [software] project that [...] is disposable, e.g., a Minimum Viable Product (MVP)"* – Student ST3 |
| B2 | Tight time to deliver software products | *"[I would not use a DPD tool in] a [software] project that must be quickly completed [...], e.g., a Minimum Viable Product (MVP)"* – Student ST3 |
| B3 | Conflicting perceptions on the usefulness of a tool | *"[Using a DPD would depend on the] agreement of the people involved in the [software] project"* – Student ST12 |
| B4 | Costs derived from the tool adoption | *"[I would not use DPD] tools that are heavy [to run]"* – Senior developer SD2 |
| B5 | Poorly documented tool | *"[I would not use a DPD tool due to] lack of documentation"* – Student ST14 |
| B6 | Low precision and recall in the detection of design pattern instances | *"[I would not use DPD tools in case I was] not sure if the tools are good (i.e., high precision) for design pattern detection"* – Student ST1 |
| B7 | Little or no perceived support to understanding the internal structure or execution of a software system | *"If this tool prevents me from learning somehow, then I would not use it"* – Student ST11 |
| B8 | Inherent difficulty of adopting a tool | *"[I would not use a DPD tool due to its] learning curve"* – Student ST16 |
| B9 | Development scenario in which using the tool is inadequate or unnecessary | *"not being the proper scenario [would prevent me from using a DPD tool]"* – Student ST13 |
| B10 | Working environment does not accommodate the use of a DPD tool | *"incompatibility [of using a tool] with [the] working environment [would prevent me from using a DPD tool"* – Senior developer SD3 |

Finally, six senior developer and five student developer mentions were associated with **Tool usage limitations** as a general barrier. This barrier refers to limitations that emerge while using a DPD tool, and it includes: **Difficulty to use the tool** (B8), mentioned by four senior developers and four students; **Inappropriate tool use** (B9), mentioned by one student; and **Incompatibility with workflow** (B10), mentioned by two senior developers. The high concentration of mentions to B8 must be stressed because, in our previous work (Moreira et al, 2022), we report our own difficulty with configuring and running existing DPD tools. Such difficulty should be definitely addressed by tool designers to promote the adoption of their tools.

## 4.4 Design patterns worth detecting

**Survey question U8:** We asked potential tool users to list design patterns they considering to be worth detecting via a DPD tool. Several participants reported specific design patterns or pattern families. Most examples still refer to GoF patterns (Gamma et al, 1995), including Observer, Strategy, Decorator, Singleton, Composite, Visitor, Chain of Responsibility, Command, Abstract Factory, Factory Method, State, and Adapter. One senior developer also mentioned Dependency Injection, suggesting that non-GoF patterns may appear in participants' expectations even though GoF patterns remain salient. This concentration on GoF patterns deserves a cautious interpretation because it may reflect at least three non-exclusive factors. First, it may reflect an educational bias: many participants were students, and GoF patterns are commonly used as the standard introductory catalog of object-oriented design patterns. Second, it may reflect a tool-availability or research-tradition bias, since many existing DPD tools and studies have historically focused on GoF patterns. Third, actual industrial relevance is also plausible because GoF patterns remain widely discussed and used in object-oriented software development. Therefore, the responses indicate the salience of GoF patterns among the respondents, not necessarily the irrelevance of non-GoF or language-specific patterns.

Figure 11 summarizes the reasons that potential tool users mentioned in their responses for automating the detection (via DPD tool) of specific design patterns. We found a total of seven concrete reasons, which we depict with blue rectangles with continuous borders. The white rectangle with dashed borders represents a general reason. We annotated the reasons with two different frequencies: triangles represent the frequency of senior developers who reported a specific reason, while diamonds represent the frequency of students who reported that reason. We discuss below our major findings.

Table 8 describes and exemplifies with quotes each reported reason. Five concrete reasons address different concerns and, therefore, were not grouped into general reasons: **Applicability to specific use scenarios** (R1), mentioned by three senior developers and one student; **Difficulty of manual detection** (R2), mentioned by one student; **Frequent use** (R3), mentioned by one student; **Potential threats to program execution** (R4), mentioned by one student; and **Supporting software flexibility** (R5), mentioned by one senior developer. Additionally, **Software maintenance** is the only general reason and it received mentions from two senior

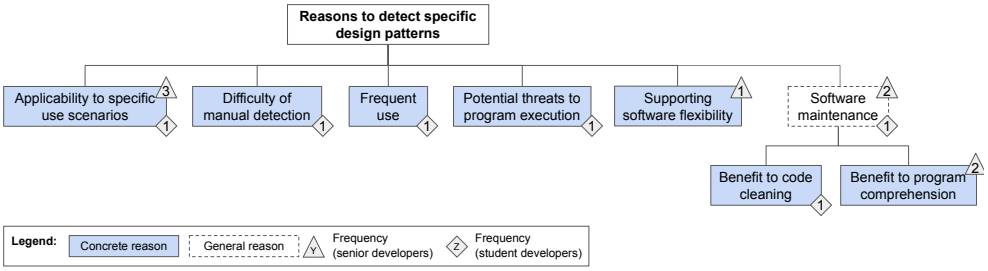



**Fig. 11**: Coding-based categorization of reasons for detecting specific patterns

developers and one student. This general reason covers two concrete reasons: **Benefit to code cleaning** (R6), mentioned by one student, and **Benefit to program comprehension** (R7), mentioned by two senior developers.

**Table 8**: Details of reasons why detecting specific patterns is worthwhile

| ID | Description | Sample quote |
|---|---|---|
| R1 | Implementing a design pattern fits a specific development context | *"if it's meaningful in the situation and the context that I am [working on], then it's worth [detecting a design pattern through a tool]"* – Student ST11 |
| R2 | Manually detecting a design pattern is challenging if not unfeasible | *"[These design patterns] are useful but may be hard to [manually] identify"* – Senior developer SD3 |
| R3 | A design pattern is frequently implemented in a given system | *"because I use [these design patterns] more often"* – Student ST3 |
| R4 | Implementing a design pattern may harm the program execution | *"[This design pattern] may cause problems in larger programs due to the use of threads"* – Student ST12 |
| R5 | Implementing a design pattern supports software flexibility | *"Structural design patterns [...] help to keep the software flexible"* – Senior developer SD5 |
| R6 | Implementing a design pattern favors code cleaning | *"[These design patterns] contribute to a cleaner code"* – Student ST3 |
| R7 | Implementing a design pattern helps understand the code structure and software execution | *"[This design pattern] would facilitate the comprehension of other developers' code"* – Senior developer SD2 |

# 5 Study implications

**Implication 1:** *Technical support may help expand the coverage of existing DPD tools* – Responses from our first survey suggest that participating tool designers seek to propose DPD tools that can be perceived as useful in industry settings. Mentions to covering multiple categories of design patterns (Figure 3), as well as the focus on popular programming languages (Figure 5), support this claim. Still, survey responses

also suggest an opportunistic choice for design patterns to detect based on technical limitations. Examples of limitations are the inherent complexity of detecting certain patterns and the availability of technologies to manipulate source code implemented in a given language. It could be that such lack of technical support is preventing some tool designers from covering more patterns and languages with their tools. Thus, we suggest that researchers and practitioners join forces to address these technical limitations in future work.

**Implication 2:** *Detection accuracy should be treated as a priority while proposing DPD tools, alongside the variety of detectable design patterns* – Some responses to our first survey (Figure 3) explicitly mention the GoF's book (Gamma et al, 1995) as a reference to design patterns implementation. Thus, some participating tool designers could justify their interest in completing their tools to detect as many GoF patterns as possible (Figure 4). The fact that users also named only GoF patterns should be read together with the possible educational, tool-availability, and industrial-relevance explanations discussed in Section 4.4. However, responses to the second survey stress that DPD is worthwhile, e.g., when a specific design pattern is frequently implemented in their systems and difficult to manually detect (Figure 11). Moreover, potential tool users mentioned that low detection accuracy may prevent them from using a tool (Figure 10). This result is quite interesting because, as our previous work revealed (Moreira et al, 2022), the accuracy of existing DPD tools is often low or insufficient. Our results suggest that detection accuracy should receive priority attention in the proposal of novel tools, while still considering pattern coverage. Section 6 of our previous work (Moreira et al, 2022) provides hints on how to address at least partially the existing accuracy issues (precision specifically).

**Implication 3:** *There may be a demand for DPD tools compatible with popular programming languages to be addressed* – Several participating tool designers expressed interest in supporting DPD on systems implemented in very popular programming languages such as C# and Python (Section 3.2). Curiously, our literature review (Moreira et al, 2022) catalogs only three tools compatible with at least one of these languages, e.g., Philippow et al. (Philippow et al, 2005) for C++ systems. More critically, none of these tools were publicly available for download during the execution of our previous work (Moreira et al, 2022). Together, these responses and our previous literature review suggest that there may be an unmet demand for tools targeting popular languages. Future work could investigate and address such demand in industry.

**Implication 4:** *The proposal of DPD tools should consider that DPD is only part of a broader software development life cycle* – Some participating tool designers reported awareness of their tools being used in program comprehension and integrated with code analysis infrastructures (Figure 7). This finding meets the users' perceptions of tools as facilitators of program comprehension (Figure 9). However, our second survey reveals other expected benefits of using a tool, e.g., facilitate bug fixes and the addition or enhancement of software features (Table 6). This finding suggests that DPD tools can be used with more complex intents in mind when compared to the pure program comprehension. These responses suggest that tool designers may consider this observation when proposing tools, so that DPD tools could be integrated with other

tools, e.g., for bug fixing or feature enhancement. This could improve the expected usefulness of DPD tools from the viewpoint of tool users.

**Implication 5:** *Documentation of DPD tools should be significantly improved for the sake of tool usefulness* – Our second survey revealed that the difficulty to use the tool and its incompatibility with the working environment may prevent tool users from adopting a DPD tool (Figure 10). It is important to note that this implication emerged from both student and senior developers, since difficulty to use the tool was mentioned by four participants in each group. However, this observation is critical if we consider that we as experts had difficulty to configure and run some of the tools empirically assessed in our recent work (Moreira et al, 2022). For student developers, limited professional and tooling experience may still act as a confounding factor, potentially amplifying the perceived difficulty of using DPD tools. Additionally, potential tool users mentioned that the lack of tool documentation plays an important role in discarding a DPD tool. We also reported in our previous literature review that existing DPD tools are poorly documented, which makes the experience of using a DPD tool even more negative. Thus, future DPD tool work should pay careful attention to tool documentation.

# 6 Discussion

The two surveys provide complementary views of DPD tools. The survey with tool designers helps to explain why DPD tools support certain design patterns and programming languages, while the survey with potential tool users helps to clarify the contexts in which such tools may be useful, the benefits expected from them, and the barriers that may prevent their adoption. Therefore, we investigate not only whether tool designers and tool users mention similar topics, but also whether the rationale behind tool construction matches the practical expectations behind tool adoption. Table 9 summarizes the main points of convergence and observed gaps between both groups. The comparison should be read as an analytical synthesis of our qualitative findings rather than as a statistical comparison, since the two surveys involved different participant groups, different questions, and exploratory samples.

**Table 9**: Convergences and gaps between tool designers and tool users

| Topic | Convergence | Observed gap |
|---|---|---|
| Design patterns | Focus on GoF patterns | Users value practical detectability and maintenance relevance, not only catalog coverage |
| Programming languages | Popular languages are considered relevant by both sides | Designers are constrained by technical infrastructure and available resources |
| Use contexts | Program comprehension and feature enhancement appear in both perspectives | Users frame DPD tools as part of broader maintenance and evolution activities |
| Adoption | Both groups recognize technical limitations | Users emphasize documentation, accuracy, usability, cost, and workflow fit |

The comparison between tool designers and tool users suggests a partial, rather than absolute, mismatch. Both groups converge on the relevance of well known design patterns, especially GoF patterns, and on maintenance related contexts, such as program comprehension and feature enhancement. However, this convergence should be interpreted carefully. GoF patterns may appear prominently because they are widely documented, commonly taught, and frequently used as a shared vocabulary for discussing design patterns. Therefore, their dominance does not necessarily imply that non-GoF, language specific, or framework specific patterns are irrelevant. The main difference lies in how each group frames the value of DPD tools. Tool designers tend to justify tool scope through aspects such as pattern popularity, catalog coverage, pattern structure, detection feasibility, language popularity, and availability of technical resources. These are legitimate concerns for building DPD tools. Tool users, in contrast, evaluate usefulness in terms of concrete development tasks, such as feature enhancement, maintenance, program comprehension, bug fixing, and reasoning about design pattern adoption. Thus, supporting more patterns or more languages does not automatically make a DPD tool useful; detection becomes valuable when it supports practical software engineering work.

A similar issue appears in programming language support. Designers reported awareness of language popularity and practitioners' demands, but their choices are also constrained by compiler support, available resources, and prior expertise. Meanwhile, users reported daily use of languages such as JavaScript, C#, Java, and Python (Figure 1a). This observation suggests that a mismatch may emerge even when designers recognize user needs, because the languages most relevant to developers may not be the easiest ones for implementing robust DPD tools.

The clearest mismatch concerns adoption barriers. Designers mainly report constraints related to tool construction, such as pattern complexity, language infrastructure, and resource availability. Users, however, emphasize barriers related to tool adoption, including difficulty of use, lack of documentation, low detection accuracy, cost-related issues, inappropriate use scenarios, and workflow incompatibility (Figure 10). This distinction is important: a technically sophisticated DPD tool may still have limited practical impact if developers cannot easily configure it, trust its results, or integrate it into their workflow.

Overall, DPD tools should be evaluated not only by detection scope, supported languages, or accuracy, but also by their usefulness in realistic development tasks. Future tools should make detection results actionable for maintenance, comprehension, feature evolution, bug investigation, and design decision assessment. Otherwise, DPD tools may remain technically relevant but weakly aligned with developers' practical needs.

# 7 Threats to validity

We discuss below threats to the validity of our study. For this purpose, we rely on strict guidelines for experimentation in software engineering (Wohlin et al, 2012).

**Construct validity** concerns whether the operationalization of the study adequately reflects the intended theoretical constructs. We carefully defined our study

goal and RQs (Section 2.1) based on multiple iterations of meetings with all coauthors of this paper. We then expected to avoid defining a weak goal, one that could not lead to insightful findings on the design and usefulness of DPD tools. Additionally, we relied on strict literature guidelines for conducting online surveys in software engineering (Punter et al, 2003). We aimed to keep the questions short and simple, thereby reducing, but not eliminating, the risk of misunderstanding when responding to the surveys. However, the data collected in our study are self-reported, which may introduce subjectivity in participants' responses. We reviewed our survey structure (Section 2.2) in multiple discussion sessions with all coauthors of this paper. Nevertheless, we did not document a formal pilot test of the questionnaire before data collection. Therefore, some questions may have been interpreted differently by participants, and this limitation should be considered when interpreting the reported perceptions and the codes derived from open-ended answers. Although some DPD tools were proposed many years ago, publication age is not itself a threat to the exploratory goal because the study intentionally relies on a systematic catalog covering two decades of DPD tools; the specific risk is that respondents may be subject to memory biases or retrospective rationalization when reporting past design decisions. None of the survey questions has changed once we forwarded the surveys to participants.

**Internal validity** concerns whether aspects of the study execution may have influenced the data collected. To recruit as many participants as possible to our first survey (Section 3), we used the full list of authors of the 42 DPD tools found in our literature review (Moreira et al, 2022). Thus, we expected to avoid selection biases regarding the participation of DPD tool designers with specific viewpoints. We waited two months for responses so that tool designers could comfortably answer all survey questions. Nevertheless, participants may have interpreted some survey questions differently, potentially affecting their responses. Regarding our second survey (Section 4), we opted to recruit tool users via broadcasting on *WhatsApp* groups, which are popular in Brazil, and one Software Engineering class. Because the WhatsApp recruitment relied on reachable Brazilian social and professional networks rather than on a controlled sampling frame, the sample composition may have been influenced by local community structure, peer forwarding, and the kinds of developers connected to these groups. Differences in participants' backgrounds and experience levels may also have influenced how questions were understood and answered. We waited one month for responses, which was long enough at least for the students who are potential tool users.

**Conclusion validity** is associated with the data analysis procedures. As discussed in Section 2.4, we relied on strict guidelines to analyze our open survey questions (Cruzes and Dyba, 2011; Stol et al, 2016). Combined with our previous experience in analyzing quantitative and qualitative data (Nascimento et al, 2021; Oliveira et al, 2020), the literature support was essential to guide the survey data analysis. The first and second paper authors of this paper had multiple meeting sessions to discuss disagreement and reach consensus. Thus, we expected to reduce human biases in both code extraction and coding-based categorization construction. However, we did not compute quantitative inter-coder agreement measures, such as Cohen's kappa or Krippendorff's alpha, and we did not systematically report the number of disagreements resolved during consensus meetings. This limits the transparency of the coding

reliability assessment, even though disagreements were discussed among the authors until consensus was reached. For the closed questions, we simply computed descriptive statistics (Wohlin et al, 2012).

**External validity** is related to the potential generalization of our study findings. Regarding our first survey, designers of only nine of the 42 DPD tools (21.4%) responded our invitation (Section 2.3). This low response rate limits representativeness; tool age mainly helps explain recruitment difficulty and possible recall bias, not a weakness in the catalog itself. Thus, the first survey categorizations, implications, and frequencies should be read as descriptive evidence from respondents, not as population-level prevalence estimates. Across both surveys, categories with one or two mentions should be interpreted as isolated or low-frequency mentions rather than robust recurring patterns, and are reported only to preserve response diversity. Regarding the second survey, we managed to recruit 17 students and 11 senior developers. Although this sample is not balanced, it is based on convenience and shaped by Brazilian WhatsApp networks and one Software Engineering class. Since we did not characterize each dissemination group, the results should be interpreted as context-bound rather than representative of developers across countries, corporate settings, or open-source communities. Before using WhatsApp, we also contacted 200 active contributors of GitHub[3] projects over two months, but received no responses. Future replications with larger and more diverse samples are needed.

# 8 Related work

To the best of our knowledge, this paper presents the first survey-based study designed for capturing the perceptions of tool designers and tool users on DPD tools. Thus, we could not directly compare our study data with data reported by past surveys. Still, we found a few survey- or interview-based studies (Ribeiro et al, 2024; Zhang et al, 2024; Hasan and Ahammed, 2025; Sharma et al, 2023; Yu et al, 2024; Gan et al, 2025; Pomian et al, 2024; Ciniselli et al, 2023) on the expected usefulness of software development tools other than DPD tools (which is similar to the purpose of our $RQ_2$)). As discussed in Section 2.1, these studies stress the need for understanding an eventual mismatch between the viewpoints of tool designers and tool users. After discussing some empirical studies on design patterns (Nazar et al, 2022, 2023), we also draw parallels between the discussions provided in our paper and those provided by previous survey studies.

Recent papers have proposed and evaluated DPD tools (Mzid et al, 2024; Nazar et al, 2022, 2023). We have not invited the designers of these recently proposed DPD tools to participate in our survey because their respective papers were published after our reference literature review (Moreira et al, 2022). However, these related papers highlight recent trends in design pattern detection. For instance, Nazar et al (2022) proposed a design pattern detection tool that creates a semantic representation of Java source code using the code features. It then applied Word2Vec to construct the word-space geometric representation of the source code. In their evaluation, the proposed tool outperformed two previous DPD tools: FeatureMaps and MARPLE-DPD.

[3] https://github.com/

In another work, Nazar et al (2023) developed an online tool, named CodeLabeller, to provide an approach for labeling Java source code files with their respective design patterns. In a survey with 25 design pattern experts, participants considered the tool easy to use after annotating the corpus of source code files for supervised machine learning classifiers. Unlike these related works, we do not propose a DPD tools although our results may be considered useful for future tool proposals.

Code and test smells research has explored developers' perceptions and expectations towards software implemented for detecting and managing such problems. For instance, Ribeiro et al (2024) presented a qualitative interview study with 7 industry developers on practitioners' perceptions of code smells and the use of code smell detection tools. Their findings suggest that code smells are important, developers understand their significance, and employ detection tools in this context. However, adoption is constrained by the configuration effort, organizational culture, and low priority level. In a similar direction, both Hasan and Ahammed (2025) and Sharma et al (2023) discussed the connections between developers' perceptions and empirical evidence in the context of quality problems in tests. The results of these studies suggest that developers perceive some testing-related issues to be particularly troublesome in some cases, even if empirical assessments do not always support that perspective, showing that there is a potential misalignment between practitioner views and evidence-based knowledge in terms of empirical verification.

Focusing on survey and interview studies with software developers, one close related work was performed by Zhang et al (2024). This empirical study collected evidence by interviewing 10 software experts and surveying 310 software practitioners. Their goal was to examine developers' intentions for code smell detection tools and the degree of alignment with current practice. This study indicated that practitioners demand tools that are context-aware, easy to use, efficient, and practically useful, while development work focuses largely on examining detection accuracy. Our results confirm the misalignment between practitioners' expectations and tool designers also observed by Zhang et al (2024). Furthermore, although we were inspired and followed the same spirit and motivation of Zhang et al (2024) work, we focus on design patters while their work investigated code smell detection.

In the context of software testing, Yu et al (2024) also investigated practitioners' beliefs about test case generation. Their work explored developers' expectations regarding automated test generating tools using 13 practitioners and a survey of 339 practitioners. Results reveal some developers have concerns regarding the existing tools such as complexity of handling scenarios, multiple platform capabilities and reliability of tests, indicating a gap between what researchers propose and what practitioners need. In another study, Gan et al (2025) explored developers' perceptions regarding integrating UI testing frameworks into CI/CD pipelines in open-source projects. This work included a survey with 94 developers and interviews with 18 practitioners. The results list several major challenges, including flaky tests, environment inconsistencies, slow execution, and difficulties in defining testing strategies.

Both Pomian et al (2024) and Ciniselli et al (2023) examined developers' perspectives on software development tools designed to support programming tasks, stressing the alignment of the tool capabilities with what users require and expectations. Pomian

et al (2024) blended LLM ideas and IDE capabilities to improve refactoring support with static analysis and IDE options. The strategy was validated through surveys with 16 industrial developers. On a different direction, Ciniselli et al (2023) explored practitioners' expectations for code recommendation systems through a survey with 80 developers. Results of their study include a taxonomy of 70 requirements that should be considered when designing code recommender systems. Our work also presents a coding-based categorization to be considered by designers of DPD tools. Furthermore, similar to our work, their findings illustrate the need for the consideration of practitioner feedback when creating developer-supporting tools.

Overall, previous studies have demonstrated the critical role of developers' attitudes and desires in designing software-based resources and that they frequently experience issues with the correspondence between research output and practitioner requirements. Our study has deepened this scope of investigation, concentrating on DPD tools, taking into account the viewpoints of both tool developers and potential tool users. Our findings reveal coding-based categorizations of contexts in which developers could use these tools (Figure 8), anticipated advantages (Figure 9), and obstacles (Figure 10). Furthermore, we present a coding-based categorization of reasons why identifying certain design patterns is useful (Figure 11), supporting exploratory knowledge for designing future DPD tools to better fit practitioner expectations.

# 9 Final remarks

This paper introduced an empirical study aimed at capturing the perceptions of tool designers and potential tool users on DPD tools. We carefully conducted two online surveys based on strict literature guidelines (Punter et al, 2003). Our first survey (Section 3) invited designers of 42 DPD tools and obtained responses from designers of nine tools, providing initial evidence about the rationale behind the proposal of DPD tools to detect specific design patterns in specific programming languages. Our second survey (Section 4) targeted both students and senior developers who are potential tool users. We computed descriptive statistics and relied on strict guidelines (Cruzes and Dyba, 2011; Stol et al, 2016) for performing open coding and the generation of coding-based categorizations.

Taken together, the two surveys provided complementary but non-generalizable evidence about a partial mismatch between DPD tool construction and DPD tool adoption. Participating tool designers and potential tool users converged on the relevance of well known design patterns and maintenance-related contexts, especially program comprehension and feature enhancement. The prominence of GoF patterns should be interpreted cautiously, since it may reflect education, documentation, and shared vocabulary as much as industrial relevance. However, designers tended to justify tool scope through pattern coverage, detection feasibility, language popularity, and technical resources, whereas users evaluated usefulness through concrete development tasks, such as maintenance, bug investigation, feature evolution, and reasoning about design decisions. Thus, supporting more patterns or more programming languages is not sufficient by itself; DPD tools become useful when detection results are actionable in realistic software engineering work.

The most visible gap concerns adoption barriers. Even technically sophisticated DPD tools may have limited practical impact if developers cannot configure them, trust their results, or integrate them into their workflows. Therefore, future DPD tools should be evaluated not only by detection scope, supported languages, or accuracy, but also by documentation quality, usability, workflow fit, and support for practical maintenance and evolution tasks. As future work, we also plan to revisit the structure of design patterns reported by tool users as relevant and investigate detection strategies that improve both accuracy and practical usefulness not only of GoF patterns but also of other software patterns (Figueiredo et al, 2009).

# Statements and declarations

***Author Contributions:***

Rodrigo Moreira and Eduardo Fernandes contributed equally to this work. Eduardo Figueiredo and Filipe Fernandes contributed to data analysis, results discussion, and manuscript revision.

***Funding:***

This research was partially supported by Brazilian funding agencies: CAPES, CNPq (Grant 406089/2025-6), and FAPEMIG (Grant APQ-01488-24).

***Competing Interests:***

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

***Ethics Approval and Consent to Participate:***

The authors declare that the methods were carried out in accordance with the relevant guidelines and regulations (see Section 2.2 for details). Informed consent was obtained from all participants under data anonymity agreement. None of the experimental protocols had to be approved by the authors' institutions.

***Data Availability:***

The anonymized data used in this study, as well as other relevant study artifacts (e.g., the survey templates), is publicly available in our study replication package: https://doi.org/10.5281/zenodo.20634953

# References

Ahmad A, Waseem M, Liang P, et al (2023) Towards human-bot collaborative software architecting with chatgpt. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE). Association for Computing Machinery, New York, NY, USA, p 279–285, https://doi.org/10.1145/3593434.3593468

Ahmed I, Mannan U, Gopinath R, et al (2015) An empirical study of design degradation: How software projects get worse over time. In: Proceedings of the 9th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp 1–10, https://doi.org/10.1109/ESEM.2015.7321186

Alkharabsheh K, Crespo Y, Manso E, et al (2019) Software design smell detection: A systematic mapping study. Software Quality Journal 27(3):1069–1148. https://doi.org/10.1007/s11219-018-9424-8

Basili V, Rombach D (1988) The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering (TSE) 14(6):758–773. https://doi.org/10.1109/32.6156

Ciniselli M, Pascarella L, Aghajani E, et al (2023) Source code recommender systems: The practitioners' perspective. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp 2161–2172, https://doi.org/10.1109/ICSE48619.2023.00182

Cruzes D, Dyba T (2011) Recommended steps for thematic synthesis in software engineering. In: Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp 275–284, https://doi.org/10.1109/ESEM.2011.36

Fernandes E, Chávez A, Garcia A, et al (2020) Refactoring effect on internal quality attributes: What haven't they told you yet? Information and Software Technology 126:106347. https://doi.org/10.1016/j.infsof.2020.106347

Figueiredo E, Silva B, Sant'Anna C, et al (2009) Crosscutting patterns and design stability: An exploratory analysis. In: IEEE 17th International Conference on Program Comprehension (ICPC), IEEE, pp 138–147

Forward A, Lethbridge T (2002) The relevance of software documentation, tools and technologies: A survey. In: Proceedings of the 2nd Symposium on Document Engineering (DocEng), pp 26–33, https://doi.org/10.1145/585058.585065

Gamma E, Helm R, Johnson R, et al (1995) Design Patterns: Elements of Reusable Object-oriented Software, 1st edn. Addison-Wesley Professional, Boston

Gan X, Liang H, Brown C (2025) Challenges, strategies, and impacts: A qualitative study on ui testing in ci/cd processes from github developers' perspectives. In: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), pp 186–197, https://doi.org/10.1109/ICST62969.2025.10988972

Garcia A, Sant'Anna C, Figueiredo E, et al (2005) Modularizing design patterns with aspects: a quantitative study. In: Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD), pp 3–14

Hasan MA, Ahammed T (2025) Prioritizing test smells: An empirical evaluation of quality metrics and developer perceptions. In: 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 815–820, https://doi.org/10.1109/ICSME64153.2025.00085

Johnson B, Song Y, Murphy-Hill E, et al (2013) Why don't software developers use static analysis tools to find bugs? In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp 672–681, https://doi.org/10.1109/ICSE.2013.6606613

Kim DK (2025) Comparative analysis of design pattern implementation validity in llm-based code refactoring. Journal of Systems and Software 230:112519. https://doi.org/https://doi.org/10.1016/j.jss.2025.112519

Kim J, Batory D, Dig D, et al (2016) Improving refactoring speed by 10x. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), pp 1145–1156, https://doi.org/10.1145/2884781.2884802

Liebel G, Badreddin O, Heldal R (2017) Model driven software engineering in education: A multi-case study on perception of tools and UML. In: Proceedings of the 30th Conference on Software Engineering Education and Training (CSEE&T), pp 124–133, https://doi.org/10.1109/CSEET.2017.29

Moha N, Guéhéneuc YG (2007) PTIDEJ and DECOR: Identification of design patterns and design defects. In: Proceedings of the 22nd Conference on Object-Oriented Programming Systems and Applications (OOPSLA), pp 868–869, https://doi.org/10.1145/1297846.1297930

Molléri JS, Petersen K, Mendes E (2016) Survey guidelines in software engineering: An annotated review. In: Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement, pp 1–6

Moreira R, Fernandes E, Figueiredo E (2022) Review-based comparison of design pattern detection tools. In: Proceedings of the 29th International Conference on Pattern Languages of Programs (PLoP), pp 17:1–17:16, https://doi.org/10.5555/3631672.3631693

Mzid R, Rezgui I, Ziadi T (2024) Attention-based method for design pattern detection. In: European Conference on Software Architecture (ECSA), Springer, pp 86–101

Nascimento R, Figueiredo E, Hora A (2021) JavaScript API deprecation landscape: A survey and mining study. IEEE Software 39(3):96–105. https://doi.org/10.1109/MS.2021.3103134

Nazar N, Aleti A, Zheng Y (2022) Feature-based software design pattern detection. Journal of Systems and Software 185:111179

Nazar N, Chen N, Chong CY (2023) Codelabeller: A web-based code annotation tool for java design patterns and summaries. International Journal of Software Engineering and Knowledge Engineering 33(07):993–1009

Nguyen-Duc A, Cabrero-Daniel B, Przybylek A, et al (2025) Generative artificial intelligence for software engineering—a research agenda. Software: Practice and Experience 55(11):1806–1843. https://doi.org/https://doi.org/10.1002/spe.70005

Oliveira E, Fernandes E, Steinmacher I, et al (2020) Code and commit metrics of developer productivity: A study on team leaders perceptions. Empirical Software Engineering (EMSE) 25(4):2519–2549. https://doi.org/10.1007/s10664-020-09820-z

Olsen JK, Fox A (2019) Usage of hints on coding-based summative assessments. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp 839–844

Paixao M, Krinke J, Han D, et al (2019) The impact of code review on architectural changes. IEEE Transactions of Software Engineering (TSE) 47(5):1041–1059. https://doi.org/10.1109/TSE.2019.2912113

Palomba F, Zanoni M, Fontana F, et al (2016) Smells like teen spirit: Improving bug prediction performance using the intensity of code smells. In: Proceedings of the 32nd International Conference on Software Maintenance and Evolution (ICSME), pp 244–255, https://doi.org/10.1109/ICSME.2016.27

Philippow I, Streitferdt D, Riebisch M, et al (2005) An approach for reverse engineering of design patterns. Software and Systems Modeling 4(1):55–70. https://doi.org/10.1007/s10270-004-0059-9

Pomian D, Bellur A, Dilhara M, et al (2024) Next-generation refactoring: Combining llm insights and ide capabilities for extract method. In: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 275–287, https://doi.org/10.1109/ICSME58944.2024.00034

Punter T, Ciolkowski M, Freimut B, et al (2003) Conducting on-line surveys in software engineering. In: Proceedings of the 2nd International Symposium on Empirical Software Engineering (ISESE), pp 80–88, https://doi.org/10.1109/ISESE.2003.1237967

Ralph P (2019) Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Transactions on Software Engineering 45(7):712–735. https://doi.org/10.1109/TSE.2018.2796554

Ribeiro F, Fernandes E, Figueiredo E (2024) A preliminary interview study on developers' perceptions of code smell detection in industry. In: Quality of Information and Communications Technology. Springer Nature Switzerland, Cham, pp 344–352

Russo D (2024) Navigating the complexity of generative ai adoption in software engineering. ACM Transactions on Software Engineering and Methodology (TOSEM) 33(5):1–50

Santana A, Figueiredo E, Alves Pereira J, et al (2024) An exploratory evaluation of code smell agglomerations. Software Quality Journal 32(4):1375–1412

Santos G, Muzetti I, Figueiredo E (2024) Two sides of the same coin: A study on developers' perception of defects. Journal of Software: Evolution and Process 36(10):e2699

Sharma T, Georgiou S, Kechagia M, et al (2023) Investigating developers' perception on software testability and its effects. Empirical Software Engineering 28(5):120. https://doi.org/10.1007/s10664-023-10373-0, URL https://doi.org/10.1007/s10664-023-10373-0

Stol KJ, Ralph P, Fitzgerald B (2016) Grounded theory in software engineering research: A critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), pp 120–131, https://doi.org/10.1145/2884781.2884833

Wang W, Tzerpos V (2005) Design pattern detection in Eiffel systems. In: Proceedings of the 12th Working Conference on Reverse Engineering (WCRE), pp 1–10, https://doi.org/10.1109/WCRE.2005.14

Witschey J, Zielinska O, Welk A, et al (2015) Quantifying developers' adoption of security tools. In: Proceedings of Foundations of Software Engineering (FSE), pp 260–271, https://doi.org/10.1145/2786805.2786816

Wohlin C, Runeson P, Höst M, et al (2012) Experimentation in Software Engineering, 1st edn. Springer Science & Business Media, New York

Yu X, Liu L, Hu X, et al (2024) Practitioners' expectations on automated test generation. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), p 1618–1630, https://doi.org/10.1145/3650212.3680386

Zanoni M, Fontana F, Stella F (2015) On applying machine learning techniques for design pattern detection. Journal of Systems and Software (JSS) 103:102–117. https://doi.org/10.1016/j.jss.2015.01.037

Zhang Z, Yin S, Wei W, et al (2024) Practitioners' expectations on code smell detection. In: 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), pp 1324–1333, https://doi.org/10.1109/COMPSAC61105.2024.00175