The design of programming tools is slow and costly. To ease this process, we have developed a design pattern catalog aimed at providing guidance about how to design tools for developers. This guidance is grounded in Information Foraging Theory (IFT), which empirical studies have shown to be useful for understanding how developers look for information during development tasks. We have facilitated a community-based approach for collecting design patterns in the catalog by having members of the research community author patterns on a publicly visible wiki. The design patterns concretely explain how to apply IFT in tool design. We conducted three evaluations of the design pattern catalog. First, qualitative analyses revealed several strengths and weaknesses of the entire design pattern catalog, in terms of criteria like problem domain coverage, abstraction level, generalizability and interconnectivity. The qualitative analyses also revealed that the community-generated design patterns compared well in quality to patterns that we had ourselves published in a smaller, peer-reviewed catalog. Second, feedback from industrial tool designers highlighted the potential value of the design pattern catalog in practice. Third, a between-subjects experiment demonstrated that students value the guidance provided through both an online wiki and through printed materials. The successful authoring of design patterns by researchers and subsequent evaluations illustrates a process of connecting Information Foraging Theory with the day-to-day needs of tool designers.
Master of Science thesis of Md Tahmid-un Nabi presented on May 26, 2016

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Md Tahmid-un Nabi, Author
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to those who are most responsible in helping me to complete this thesis. First and foremost, I would like to thank my family for providing me with constant support. Right from my childhood, my family has set examples in front of me with regards to hard work, responsibility. They have been tough but fair, supportive yet inspirational and have played a crucial role in shaping me into the person I am today. My decision to pursue graduate studies is highly dependent on my family's encouragement to continue to push myself intellectually.

I would also like to sincerely thank my major advisor Dr. Christopher Scaffidi. I have learnt an enormous amount from him. He has continuously encouraged and challenged me, and has always been patient to address any concerns I have had. He has helped me achieve significant level of personal growth with regards to writing ability, intellectual curiosity, critical thinking and confidence.

Throughout the duration of my studies I have had the pleasure to work with a great team of researchers who have guided me and contributed to my research. They include in no particular order: Dr. Margaret Burnett, Dr. Scott Fleming, David Piorkowski, Kyle M.D. Sweeney and Sam Lichlyter. Their contribution to the progress of this thesis has been vital.

This thesis is the continuation of the work originally published in [16]. The design pattern catalog published in [16] has been expanded through a community-based process. Researchers from around the world contributed 16 additional design patterns, which has broadened and deepened the range of ideas for how to apply IFT for tool design. Additionally, the design pattern catalog has been evaluated in a variety of ways like qualitative analysis, questionnaires to professional tool designers and empirical studies assessing the usefulness of the catalog for training students.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Overview of Information Foraging Theory</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Key concepts of IFT</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Related Work leading to this project</td>
<td>7</td>
</tr>
<tr>
<td>3 Related Works</td>
<td>11</td>
</tr>
<tr>
<td>3.1 The established success of design patterns</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Methods for eliciting design pattern catalogs</td>
<td>11</td>
</tr>
<tr>
<td>3.3 The limited methods available for evaluating design pattern catalogs</td>
<td>13</td>
</tr>
<tr>
<td>4 Collection of IFT-based Design Patterns</td>
<td>16</td>
</tr>
<tr>
<td>4.1 Recruitment of Design pattern Authors</td>
<td>16</td>
</tr>
<tr>
<td>4.2 Collecting Patterns through a public Wiki</td>
<td>16</td>
</tr>
<tr>
<td>4.2.1 Getting Started</td>
<td>17</td>
</tr>
<tr>
<td>4.2.2 Registration</td>
<td>18</td>
</tr>
<tr>
<td>4.2.3 A short primer on Information Foraging Theory</td>
<td>19</td>
</tr>
<tr>
<td>4.2.4 Rules for a valid IFT-based Design Pattern</td>
<td>20</td>
</tr>
<tr>
<td>4.2.5 Design Pattern description format</td>
<td>21</td>
</tr>
<tr>
<td>4.2.6 Walkthrough</td>
<td>23</td>
</tr>
<tr>
<td>4.2.7 List of Patterns in the Pattern catalog</td>
<td>24</td>
</tr>
<tr>
<td>4.3 Protocol for Design Pattern Submission and Review</td>
<td>25</td>
</tr>
<tr>
<td>4.4 Design Patterns authored by our recruited authors</td>
<td>26</td>
</tr>
<tr>
<td>5 Evaluation Part 1: Qualitative analyses of design pattern catalog</td>
<td>30</td>
</tr>
</tbody>
</table>
## 5.1 Methodology

5.1.1 RQ1: Is there an unequal problem domain coverage by tool builders/pattern authors? .......................................................... 31

5.1.2 RQ2: How abstract are the patterns? ............................................. 32

5.1.3 RQ3: How widely used are the patterns in practice? ......................... 33

5.1.4 RQ4: How are the Design Patterns inter-connected? .......................... 34

5.1.5 RQ5: How “generalizable” are the patterns across different Software Engineering task types? .......................................................... 36

5.1.6 RQ6: Are the community-generated patterns at least as “good” as those that we ourselves wrote? ......................................................... 38

5.1.7 RQ7: What are the areas of design pattern description that pattern authors struggled with? .......................................................... 39

## 5.2 Results from qualitative analysis of the Entire Design Pattern Catalog

5.2.1 RQ1: Is there an unequal problem domain coverage by tool builders/pattern authors? .......................................................... 40

5.2.2 RQ2: How abstract are the patterns? .................................................. 42

5.2.3 RQ3: How widely used are the patterns in public? ............................... 42

5.2.4 RQ4: How are the Design Patterns inter-connected? .......................... 43

Patterns A and B have the same high-level purpose .................................. 44

One pattern is a sub-pattern of the other ................................................. 45

The patterns A and B can be used in conjunction with one another .......... 46

Pattern A can be used in a prior/subsequent step/phase before Pattern B ...... 47

Patterns A and B use the same data as input ........................................... 48
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.5 RQ5: How “generalizable” are the patterns across different Software Engineering task types?</td>
<td>49</td>
</tr>
<tr>
<td>5.3 RQ6: Are the community-generated patterns at least as “good” as those that we ourselves wrote?</td>
<td>49</td>
</tr>
<tr>
<td>5.3.1 Criteria: IFT objectives addressed</td>
<td>49</td>
</tr>
<tr>
<td>5.3.2 Criteria: Level of Abstraction</td>
<td>52</td>
</tr>
<tr>
<td>5.3.3 Criteria: Generalizability</td>
<td>53</td>
</tr>
<tr>
<td>5.3.4 Criteria: Type of Known Uses</td>
<td>54</td>
</tr>
<tr>
<td>5.4 RQ7: What are the areas of design pattern description that participants struggled with?</td>
<td>55</td>
</tr>
<tr>
<td>6 Evaluation Part 2: Assessing the usefulness of the Design Pattern catalog</td>
<td>59</td>
</tr>
<tr>
<td>6.1 Methodology</td>
<td>59</td>
</tr>
<tr>
<td>6.1.1 Obtaining Feedback from Professional Tool Designers</td>
<td>59</td>
</tr>
<tr>
<td>6.1.2 Measuring the utility of the patterns for training graduate students</td>
<td>61</td>
</tr>
<tr>
<td>6.2 Results</td>
<td>64</td>
</tr>
<tr>
<td>6.2.1 How useful do tool designers find the design pattern catalog to be?</td>
<td>64</td>
</tr>
<tr>
<td>6.2.2 How useful is the pattern catalog for training graduate students?</td>
<td>67</td>
</tr>
<tr>
<td>7. Limitations and Threats to validity</td>
<td>71</td>
</tr>
<tr>
<td>7.1 Pattern Collection</td>
<td>71</td>
</tr>
<tr>
<td>7.2 Qualitative Analysis and Practical Evaluation</td>
<td>71</td>
</tr>
<tr>
<td>8. Conclusion</td>
<td>73</td>
</tr>
<tr>
<td>Bibliography</td>
<td>75</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. An information environment consisting of information patches connected by links. The numbers associated with each link is the cost of traversing that link.</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2. IFT constructs in the context of a software development environment</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3. Getting Started page of the wiki</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4. A partial snapshot of the IFT primer on the wiki</td>
<td>19</td>
</tr>
<tr>
<td>Figure 5. The step-by-step walkthrough guide</td>
<td>24</td>
</tr>
<tr>
<td>Figure 6. Excerpt from the list of patterns in the wiki</td>
<td>25</td>
</tr>
<tr>
<td>Figure 7. Patterns per IFT objective</td>
<td>40</td>
</tr>
<tr>
<td>Figure 8. Graph showing relations between patterns in the current catalog</td>
<td>44</td>
</tr>
<tr>
<td>Figure 9. Patterns having sub-patterns</td>
<td>46</td>
</tr>
<tr>
<td>Figure 10. Patterns which can be used concurrently</td>
<td>47</td>
</tr>
<tr>
<td>Figure 11. Patterns having a sequential relationship</td>
<td>48</td>
</tr>
<tr>
<td>Figure 12. Patterns related by using same data as input</td>
<td>48</td>
</tr>
<tr>
<td>Figure 13. Patterns per IFT objective: TOSEM and community-generated</td>
<td>50</td>
</tr>
<tr>
<td>Figure 14. Objective coverage level comparison</td>
<td>51</td>
</tr>
<tr>
<td>Figure 15. Average numbers of Known Uses cited per design pattern (categorized according to whether each Known Use’s tool was industrial, was created by the pattern author’s research group, or was created by another research group)</td>
<td>54</td>
</tr>
<tr>
<td>Figure 16. Feedback count per Design Pattern section</td>
<td>55</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1: Questions to ask about a tool ................................................................. 23
Table 2: Design Patterns from our recruited authors............................................. 27
Table 3: List of IFT objectives .............................................................................. 31
Table 4: Type of Relations between Patterns ....................................................... 35
Table 5: Software Engineering Task Types .......................................................... 37
Table 6: Patterns related by same high-level purpose (note: a pattern may have more than one high-level purpose) ................................................................. 45
Table 7: Summary of Agreement with statements (1=Strongly Disagree through 5=Strongly Agree); Google Column shows agreement by industrial tool designers (Section 6.2.1) and the rightmost two columns show agreement by graduate students (Section 6.2.2) ......................................................................................... 69
Table 8: Statements about paper prototyped features between Condition 1 (Digital Wiki) and Condition 2 (Printed Known Use) .......................................................... 70
Mapping Software Development Tool Design to Information Foraging Theory

1 Introduction

Tools play a central role in enabling software developers to find information efficiently during development tasks. For example, such tools include search and recommendation functions that can help a developer find the location of a bug in order to fix it [31][32]. Other tools enable developers to discover and view bug reports relevant to a piece of code [13], to leave and view notes for one another [53], to organize work files visually in a way that minimizes effort [21], and to perform a myriad other functions aimed at easing the challenge of finding information.

However, tool designers interested in contributing new approaches face two key challenges. In the early stages of formulating of a new tool, tool designers need to define the problem that the tool should solve, as well as identify a solution to said problem. Problem definition and solution searching can involve time-consuming, unstructured, potentially fruitless iteration. Second, standing out from existing tools demands innovation, which in turn requires knowledge of the large and rapidly growing literature. Obtaining this comprehensive awareness is time-consuming and potentially a barrier to entry for novice tool designers.

To date, tool designers have relied primarily on intuition and empirical study to overcome these challenges. For example, one tool embodied the insight that developers often need to navigate through code based on what lines of code could be called during certain situations, and the tool included a novel static analysis for supporting such navigations [30]. But this insight was gleaned only after long, expensive empirical research [29].

In short, the time, the cost, the need for comprehensive knowledge of the literature, and the unstructured process of idea formulation combine together to slow the rate of tool innovation to a crawl. These considerations suggest that those interested in designing tools for developers would benefit from a synthesis of the relevant literature, in a form that guides tool innovation, highlights open challenges, and sparks new insights about how to design tools.
In prior work, our research group took the first step toward these goals by performing a literature review [16] framed by Information Foraging Theory (IFT) [45]. This theory, previously applied to web browsing behavior [12][41], explains and predicts how developers seek information [31][32][42]. The literature review examined software engineering papers and explained how programming tools reveal ways of applying IFT in practice [16]. This yielded 12 design patterns summarizing how those tools, at an abstract level, applied IFT concepts to help developers.

A key limitation of that preliminary catalog is that it only incorporated our own research group’s perspectives. There remains a need for expanding this catalog to incorporate other viewpoints and the broader understanding of the literature that can only be achieved by involving a larger research community. We also need to demonstrate that the process of representing tool insights in the form of IFT-based design patterns can be replicated by other researchers outside of our own research group.

Moreover, we have yet to demonstrate the practical value of this catalog, both to practicing and to novice tool designers. We also need to perform some form of analysis on this catalog to identify which aspects of the catalog needs improvement. Eliminating these limitations is the objective of the current work.

This thesis therefore presents an expansion of this design pattern catalog through a community-based process. Researchers from around the world contributed 16 additional design patterns, which broadened and deepened the range of ideas for how to apply IFT for tool design.

We then evaluated the design pattern catalog in 3 ways: through a qualitative analysis of the catalog against quality metrics, through a questionnaire of programming tool designers in industry, and through an empirical study assessing the usefulness of the catalog for training students.
2 Overview of Information Foraging Theory

Information Foraging Theory (IFT) is based on Optimal Foraging Theory [36]. Optimal Foraging Theory is a predictive theory about predators’ (animals) behavior when they forage for prey (food) in an environment. By applying Optimal Foraging Theory in the domain of Information Technology and equating human behavior when searching for information to predator behavior when foraging for prey, Pirolli [44] created Information Foraging Theory.

2.1 Key concepts of IFT

We present a simplified version of basic IFT constructs and propositions in a software development environment below, necessary for understanding IFT-based Design Patterns.

Information Foraging Theory aims to make predictions about human predator’s behavior when foraging for information in an information environment.

In IFT, an information environment is described in terms of a topology. This topology is made up of information patches connected together by traversable links. Figure 1 shows an example of an information environment represented in IFT.

An information patch (e.g., sections of documents or source code files, such as methods) contains information features (e.g., words, diagrams, function calls, etc.). Each information feature has a processing cost associated with it (i.e., the effort or time required to understand that piece of information). For example, in Figure 2, we can see a source code file which is serving as an information patch. Links can be used to navigate from one information patch to another.

Links connect information patches with one another. One can navigate to another information patch from the current information patch by traversing an outgoing link. Examples of links are clickable hyperlinks, clickable links between method names, menus etc. For example, in Figure 2, a clickable link (a link) to another source code file (another information patch) can be seen. Each link has an associated cost of navigation. Examples
of this cost can be time taken to traverse the link to reach one information patch from another, or effort expended by the user to traverse the link. Some information features in information patches are associated with links. These features are called *cues*.

Figure 1. An information environment consisting of information patches connected by links. The numbers associated with each link is the cost of traversing that link.

*Cues* are information features in information patches that are associated with the outgoing links from the patch. Cues provide hints about what information features can be found in the information patch the link is leading to. For example, if the link is a link from one method to another, then the name of the target method serves as a cue that tells the developer where he will go to if he clicks the link. Not all information features are cues, only those that are associated with links. For example, in Figure 2, we have a small popup
window displaying some documentation about the class which can be reached by the clickable link. Thus, this popup window is acting as a cue.

The persons who seek information in an information foraging are known in IFT as predators. Predators seek a specific set of information features. This set is called the Predator’s information goal.

The information patches in the information environment contain information features that have value in the context of the predator’s current goal. Some features may have more value than others, based on how close they are to the actual information the predator is actually foraging for.

The information features in the information environment that comprise the predator’s information goal set are termed prey.

Another concept in IFT is the concept of information scent. Information Scent refers to the predator’s perception of whether a given link will lead to desired prey. The cues associated with a link affect scent.
In Information Foraging Theory, when a predator is searching for prey in an information environment, it can take any one of the following three possible actions:

- The predator can stay in the current information patch and continue to process the information features present in the patch.
- The predator can navigate to an adjacent information patch via an outgoing link.
- The predator can add to the topology itself. This process is known as enrichment. Examples of enrichment include creating new information patches, creating a new link to an existing patch etc.

The central proposition of IFT is that predators will make choices that will maximize the estimated/expected value per estimated/expected cost of interaction. Mathematically, this is expressed by:

\[ \text{Predator's selected choice} = \max \left[ \frac{E(V)}{E(C)} \right] \]

Another proposition of IFT is that predators make the decision to navigate to a patch based on how likely the patch is to contain prey. This likelihood depends upon the cues associated with an outgoing link:

- How much the cue attracts predator’s attention
- How much the cue indicates that the prey will be found in patch the link leads to

Software developers often look for information in their development environment. Some example scenarios can be locating cause of bugs, finding references to an object/method etc. When looking for information, they incur costs as defined in IFT. Examples include:

- Cost of processing – reading and understanding code
Cost of navigation – navigating between different source files

Software developers also consider cues before making foraging decisions. Examples include:

- Think about a Class name before deciding to navigate to that class
- Read documentation about a method before trying to process it

2.2 Related Work leading to this project

Information Foraging Theory (IFT) is based on Optimal Foraging Theory [36]. Optimal Foraging Theory is a predictive theory about predators’ (animals) behavior when they forage for prey (food) in an environment. By applying Optimal Foraging Theory in the domain of Information Technology and equating human behavior when searching for information to predator behavior when foraging for prey, Pirolli [44] created Information Foraging Theory.

IFT has been proven to be extremely effective in predicting how people will seek information on the web [11][17][46][43]. Researchers have also been able to use the insights gained from IFT to design web tools which have been shown to lead to more efficient web foraging by empirical studies [12][51].

In recent years, IFT has proven to be applicable in numerous areas of Software Engineering as well. The pioneering work on applying IFT to software engineering tasks was published by Lawrance et al. [31][33], who showed that IFT constructs can be used to successfully model developer behavior during software maintenance tasks.

Lawrance et al. also presented a theory of programmer navigation during debugging based on information foraging theory in [34]. They presented a model that predicts programmer’s navigation choices when debugging source code in response to a bug report. The original information foraging constructs were refined in the context of debugging, and operationalized to make the theory applicable to real world debugging. The executable model, termed PFIS (Programmer Flow by Information Scent) computes information scent and propagates the computed scent throughout the foraging environment to make predictions about developers’ navigations. An approximation of the information Scent was
computed by computing the word similarity between the bug report and source code using a vector space IR model. The propagation of the computed information scent was achieved by using the spreading activation technique [1]. An empirical study mentioned in the paper showed that the PFIS model was more accurate at predicting programmer navigation behavior than other comparable models.

Lawrance et al. subsequently improved upon PFIS in a future work [35], presenting PFIS2. Compared to previous models, PFIS2 adjusts its notion of prey after each step. This allows PFIS2 to tackle real-world scenarios like modelling users’ changing goals, and the information environment changing significantly during the foraging process. In addition to a base PFIS2 model, several variants were also proposed in the paper.

Piorkowski et al. performed an empirical study evaluating the performance of various predictive models [42] modelling developer navigation. They also created multi-factor predictive models by combining single factors into PFIS3 using a spreading-activation based approach, and empirically evaluated these multi factor models as well.

There has been prior research done on looking at end user programmers’ behavior during debugging from an IFT perspective as well. Kuttal et al. [27] studied end users’ foraging behavior during debugging web mashups and presented a model based on IFT to explain such behavior. They conducted a “think-aloud” empirical study in which they had participants (end users) complete a set of debugging tasks. They then interpret the user behavior through an IFT lens. Their presented model of foraging behavior consists of two main phases: Fault Localization and Fault Correction. They also present a categorization of the various types of cues that are present in a web mashup programming environment.

Niu et al. investigated the role IFT can play in code navigation tools [39]. They conducted an exploratory study aimed at uncovering a more complete patch-model which can offer a unified account of how programming tools can assist in code navigation tasks.

Niu et al. also leveraged IFT to develop a foraging model for a requirements tracing tool [40]. In the study, they also compared and contrasted the behavior of real analysts with
that of the optimal information forager as predicted by the model. The results of this study offered insights into potential opportunities for improving tool support in such tasks.

A recent work by Bhowmik et al. [5] looked at information foraging of developers in social groups. They looked at the process of developers working together to resolve tasks in open source software projects from a social information foraging perspective and used this to predict optimal group size. They also analyzed effect of group size on individual performance in such projects.

In addition to using IFT to predict and model developers’ behavior during software engineering tasks, IFT can also be used to obtain insights into how and why software engineering tools can aid developers. Fleming et al.’s work in [16] looks at software engineering tools from an IFT perspective. They at first discuss the applicability of IFT to information-intensive software engineering tasks. The key contribution of this paper is identifying and documenting twelve recurring IFT-based design patterns in successfully software engineering tools.

Thus, IFT can easily model software developer’s interactions with the development environment. As already discussed in the above, prior empirical studies have extensively validated the benefits of applying IFT to software engineering. Fleming et al.’s work in [16] looks at software engineering tools from an IFT perspective. They at first discuss the applicability of IFT to information-intensive software engineering tasks. The key contribution of this paper was in identifying and documenting twelve recurring IFT-based design patterns in successfully software engineering tools. For example, one of the twelve design patterns described in the paper was the Dashboard pattern, which refers to an information patch in which a developer can become aware of links that lead to continually changing information patches that have high value. For instance, a developer might need to manage many different tasks, and rather than having to check on each task periodically by navigating to and inspecting task-relevant artifacts, the Dashboard provides a single patch that provides up-to-date task information. Multiple tools, such as the popular Bugzilla bug-tracking system, illustrate this design pattern in action.
This thesis is a continuation of the work of Fleming et al. Our study collects additional IFT-based design patterns from software engineering research through a community process (a public wiki). We also analyze the IFT-based design patterns to gain additional insights into the current design pattern and tool design landscape.
3 Related Works

3.1 The established success of design patterns

Design patterns are typically defined as general, reusable solutions to common design problems [8][19]. Design Patterns have had numerous applications in industry including being used for designing APIs [7], service-oriented architectures [14] and object-oriented systems [4][19][58]. Design patterns have proven to be useful in software engineering research as well. Researchers have leveraged design patterns in designing embedded systems [2][15], tools for visualization [22], dynamically adaptive systems [48] and software for synchronous collaboration [23]. Design patterns have also been found to be useful in several other research areas, such as in computer security [56], human computer interaction [9][26], game design, multi-agent systems [24] and computer-supported collaborative learning [3]. The usefulness of design patterns can be attributed to their level of abstraction and concreteness. The level of abstraction of standardized design patterns ensure that solutions are generalizable across multiple contexts [18][19][38][49]. However, studies have also shown that design patterns are also concrete enough to enable designers create more maintainable designs [47]. Even though design patterns have been applied to many areas of software engineering, currently no catalog of patterns exist that can be used to design tools supporting developers’ navigations during software maintenance. Our effort to map IFT constructs and propositions to design patterns for software engineering tools addresses this gap.

3.2 Methods for eliciting design pattern catalogs

Researchers have also looked at various methods for eliciting design patterns in various areas. Baggetun et al. [3] discussed several possible approaches for eliciting design patterns in the area of computer-supported collaborative learning. According to them, there are two main approaches for eliciting design patterns:

- Deductive processes – begin with general views and move toward specific ones
Inductive processes – begin with specific views and move toward general ones

Baggetun et al. also gave examples of various inductive and deductive processes. Examples of inductive processes include deriving patterns from instances, human behaviors, and interrelations. Examples of deductive processes include deriving patterns from metaphors, mindmaps, and experiences.

In [55], Winters et al. provide additional examples of both inductive and deductive approaches to design pattern elicitation. In Winters et al.’s work, inductive methods for eliciting design patterns include:

- Ad-hoc discussion – often applied in computer gaming [25].
- Multi-disciplinary description and validation – Leo et al. [28] developed patterns for collaborative learning by having patterns identified and described by collaborative learning practitioners, and then validated by pedagogy experts.
- A systematic pattern development cycle – patterns are identified by reverse engineering systems which embed good design. After identification, patterns are interrelated to form a pattern language [54].

Deductive methods for eliciting design patterns include:

- Workshops – participants present patterns they are working on in a workshop and receive feedback. A prominent example of this approach are the writer’s workshops held at the annual Pattern Languages of Programs (PLoP) conference.
- Shepherding – Design patterns authored by pattern developers are commented on by other pattern developers. A description of the process can be found in Harrison’s work [20].
• Civic participation – An example of this would be Schuler’s work [50], in which patterns were solicited by an open call circulated to a range of mailing lists.

Our study can be said to be a composite of civic participation and shepherding. We solicit design patterns from software engineering researchers by requesting them to document their design patterns on a community wiki. However, we also shepherd our pattern authors to improve and finalize their design patterns by providing continuous feedback to their work.

3.3 The limited methods available for evaluating design pattern catalogs

There have been multiple research works aimed at analysis of design pattern catalogs in various domains. Yskout et al.’s work [56] analyzed the existing patterns in software security engineering. Their aim was to provide a better understanding of the current landscape of patterns and also to develop a framework to identify relevant patterns that could be implemented as a concrete software solution. They aimed at answering the following questions about the existing security patterns:

• Are there enough patterns?
• Are all patterns constructible software solutions?
• Are the patterns properly documented?
• Are the patterns useful in practice?

From their analysis, Yskout et al. obtained several insights. One insight was that although problem domain coverage by security patterns were not insufficient, the coverage was not uniform. They also came to the conclusion that quality of documenting security patterns needed significant improvement. Another insight from their work was that only in few cases did the patterns showed themselves useful as solutions in real-life systems.

Several of our research questions aimed at evaluating the quality of our design pattern catalog are based upon the questions asked by Yskout et al. We also adapt some of
their coding schemes for security patterns to apply to our IFT-based design patterns. More details about this can be found in the Methodology section.

In [24], Juziuk et al. performed a systematic literature review of existing design patterns for multi-agent systems. They had a systematic protocol for determining whether a study met the criteria for inclusion in the review. They only included those studies which concerned design patterns for multi-agent systems, were published between 1998 and 2012, and were written in English. However, if the patterns in the study were not described in detail or were improperly structured, or a newer study containing the same patterns existed, or the study itself was a review, it was excluded. Their search strategy was also systematic, with both a manual and an automatic component. The manual search strategy consisted of a manual search of the *International Journal of Agent-Oriented Software Engineering* (IJAOSE). The automatic search consisted of a search based on a list of keywords in electronic databases like ACM Digital Library.

After identifying relevant literature, Juziuk et al. aimed at answering the following research questions:

- How were the patterns documented and what pattern templates were used?
- How were the design patterns interconnected?
- For what types of systems have the design patterns been applied?
- How can the design patterns be classified?

From their analysis, Juziuk et al. obtained several insights. One insight was that at the time of the study, no standard template for describing multi-agent system design patterns existed. To alleviate this problem, Juziuk et al. presented a potential standard template for describing future patterns. Another insight was that when a graph was created from “Related Patterns” data to find pattern interconnections, 3 clusters were found. One cluster contained object-oriented and concurrency patterns, the second cluster contained patterns from bio-inspired concepts, and the third cluster contained patterns related to mobile agents.
Our work features several similarities to the work of Juziuk et al. Although we did not conduct a systematic literature review for this study, we did follow a similar systematic approach (using a keywords-based search in electronic databases) for identifying eligible design pattern authors for our study. We also aimed at answering some of the questions Juziuk et al. asked about the multi-agent system design patterns for our own IFT-based design patterns catalog.

It is also interesting to note that both Yskout et al. and Juziuk et al. concluded that the biggest issue with security design patterns and multi-agent system design patterns respectively was lack of a standard format for documenting patterns. We have avoided this issue when collecting the IFT-based Design Patterns. We had already developed a standardized design pattern description format (described in detail in Methodology section) before beginning the pattern collection process and strictly imposed this format on all pattern authors. So, all patterns have been documented using the same standardized template and future pattern collections will continue this format.
4 Collection of IFT-based Design Patterns

4.1 Recruitment of Design pattern Authors

To identify people we could recruit as authors of design patterns, we used ACM Digital Library (dl.acm.org) to search through relevant conference proceedings (e.g. ICSE, FSE/ESEC, ASE and ICSME) for papers that describe a software engineering “tool”. We used keywords related to software engineering tasks (e.g., “tool” in combination with “maintenance”) to discover relevant literature.

We manually reviewed the search results from these keywords to confirm their relevance. We then searched the web for the current contact information of each author who was not a graduate student at the time of searching, and used this information to construct a recruitment list.

To complete the recruitment process, we sent emails to potential authors containing an attached recruitment text and a copy of the consent form, inviting authors to respond by email for further information if they were interested in participating.

When authors replied to our recruitment email, we answered any questions they had, and provided them with a consent form. They were asked if they agreed to participate in our study, and if so to confirm by email. Due to the online nature of this study, we did not require pattern authors to provide a signed consent document. However, we did save a copy of the reply email as evidence of the authors’ informed consent.

4.2 Collecting Patterns through a public Wiki

To facilitate collection of design patterns from our design pattern authors, we employed a publicly visible wiki.

The structure of the wiki is as follows:

- A Getting Started page, summarizing everything about the content and purpose of the wiki
- A registration page, allowing potential design pattern authors to sign up for the wiki
• A short primer on Information Foraging Theory, intended to provide potential pattern authors with some background knowledge on IFT that they can leverage when trying to author patterns
• A page defining the rules for a valid IFT-based Design Pattern
• A page describing the structure/template of a Design Pattern description
• A guide that walks potential pattern authors through the uncovering and description process of an IFT-based design pattern
• A page listing all the design patterns in the wiki
• A page containing description of the wiki syntax and finally,
• A Pattern template generator that could be used to generate the body of an empty pattern which can be subsequently filled in to complete the pattern description

Each of these components of the wiki are briefly described below.

4.2.1 Getting Started

The Getting Started page of the wiki is intended to be a page that provides a succinct summary of all content and functionality available in the wiki. Thus, it contains a brief description of all other pages in the wiki, along the hyperlinks to these pages. The content of the Getting Started page is structured in a way to provide a logical progression for a potential contributor to at first gain an understanding of IFT-based design patterns, and then gain insight into how to author one and submit it to the wiki itself.
4.2.2 Registration

This purpose of this page is to provide a simple form where potential design pattern authors entered their desired user name and password and complete the registration of their account. Account registration is necessary because we do not allow unregistered users of the wiki to make any edits. This is done to preserve the integrity of the wiki content.
4.2.3 A short primer on Information Foraging Theory

A certain level of understanding of Information Foraging Theory is necessary for properly authoring an IFT-based design pattern. To provide potential pattern authors with this knowledge, a short primer on Information Foraging Theory was included in the wiki. The primer briefly describes all Information Foraging constructs with relevant examples, and also provides a simplified description of the major propositions of Information Foraging Theory.

Figure 4: A partial snapshot of the IFT primer on the wiki
4.2.4 Rules for a valid IFT-based Design Pattern

To ensure integrity of the design pattern, we enforce a few rules all design patterns have to follow. These rules are:

- The pattern must be related to an activity that is somehow related to developers foraging for information in the development environment
- The pattern description must contain the sections Intent, Motivating Example, Description, Applicability, Connection to IFT, Consequences – Benefits, Consequences – Liabilities, Known Uses, Related Patterns
- Each individual section of a Pattern must also match requirements for that particular section
- For something to be a design pattern, it has to be implemented in at least 3 tools. These tools are to be described in the Known Uses section
- When mentioning the tools, it should be clear that:
  - The tool has the same intent as that of the design pattern
  - The input to the tool can be categorized as belonging to the same category of the inputs specified in the pattern description
  - The output of the tool can be categorized as belonging to the same category of the outputs specified in the pattern description
- Valid and correct references must be provided for any tool mentioned in the Known Uses section
- Valid and logical reasoning must be provided in Related Patterns when describing how the current pattern is related to other patterns

Since we want all patterns contributed by our community authors to follow these rules, these were documented in a separate article in the wiki. Potential authors were encouraged to go through the article in the Getting Started page.
4.2.5 Design Pattern description format

One of our primary aims was to ensure that all design patterns we collect be described in one consistent, standardized format. Prior analysis on various design pattern catalogs by researchers [56][24] have shown that lack of a standardized description format is the most prevalent issue in those catalogs. To avoid this, we decided to adopt a slightly modified version of the template used for describing Design Patterns in the seminal work on design patterns by Gamma et al. [2]. In addition to all the sections used for describing a pattern in that book, we added another section, “Connection to IFT”, to our design pattern description format. As we were collecting IFT-based Design Patterns, we decided that documenting how each pattern employs the principles of IFT to achieve its desired objective is a critical and essential part of the pattern description. We provide the description format of IFT-based Design Patterns below.

<table>
<thead>
<tr>
<th><strong>&lt;Pattern Title&gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intent</strong> – a short statement that answers the following questions concisely: What does the design pattern do? What is its rationale and intent? What particular information foraging issue does it address?</td>
</tr>
<tr>
<td><strong>Motivating Example</strong> – At least one real-life scenario that illustrates an information foraging problem and how the tools implementing the design pattern could solve the problem.</td>
</tr>
<tr>
<td><strong>Description</strong> – A description of how the pattern works. Includes descriptions of what things the pattern takes in as input, how these inputs are processed by the pattern and what outputs are finally produced by the pattern.</td>
</tr>
<tr>
<td><strong>Applicability</strong> – What are the situations in which the design pattern can be applied? Includes any assumptions made and all conditions that must be met for the pattern to be applicable.</td>
</tr>
</tbody>
</table>
**Connection to IFT** – In this section one uses official IFT terms and constructs to explain how the tool is solving an information foraging problem. Some common ways maybe:

- The design pattern helps identify patches containing valuable prey
- The design pattern helps enrichment of the current topology
- The design pattern increases information scent
- The design pattern reduces cost of navigating between patches

**Consequences** – What are the trade-offs and the results of using this pattern. Further subdivided into:

- **Benefits** – Here the benefits of using the design pattern are described. Possible benefits can be but are not limited to reducing cost of foraging, enriching the topology etc.
- **Liability** – The subsection mainly describes the pitfalls/costs of misusing the pattern or implement the pattern incorrectly. It can also describe scenarios in which the pattern loses its effectiveness.

**Known uses** – Examples of the pattern found in real tools. At least 3 examples must be provided if the pattern is to be considered as a “complete” design pattern. For each example. Descriptions of how they are implementing the pattern or represent the pattern in action must be provided.

**Related Patterns** – In this section one tries to provide answers to the following questions: What design patterns are closely related to this one? What are the similarities between them and what are differences? With which other design patterns can this one be used?

Since we wanted all design pattern authors to follow this format of pattern description when they authored a design pattern, this design pattern format was documented in the wiki in the form of an article. Potential authors were encouraged to go through the article in the Getting Started page.
4.2.6 Walkthrough

We realized that only providing Rules and Description Format may not be sufficient for helping a potential design pattern author to uncover and write new design patterns. Instead, if we could provide them with an actual demonstration of uncovering and then describing a new pattern, it would provide a better starting point.

Thus, we created a step-by-step walkthrough guide. The walkthrough guide consists of a visual description of the process that was followed by one of the researchers to at first uncover and then identify the necessary information to complete all the necessary sections of a design pattern.

The walkthrough starts by assuming that the potential design pattern author is considering whether one particular tool/feature of a tool is an instance of a design pattern. It then mentions some of the questions one should ask when trying to determine whether a tool/feature of a tool is an instance of a design pattern. The list of questions asked in the walkthrough itself is shown in Table 1. This is accompanied by showing an actual example of how one of the researchers used such questions to initially uncover a design pattern from one of the features of the Eclipse IDE.

Table 1: Questions to ask about a tool

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who is the predator?</td>
</tr>
<tr>
<td>What are the information patches?</td>
</tr>
<tr>
<td>Is there any prey?</td>
</tr>
<tr>
<td>Did the tool help in the foraging?</td>
</tr>
<tr>
<td>Can the changes made by the tool be mapped to specific IFT constructs, and if so, which ones?</td>
</tr>
</tbody>
</table>

If the answers to those questions are affirmative, the guide then walks the pattern authors through the rest of the process. For identifying/extracting the necessary information to complete all the required sections of a pattern, the guide provides an example of set of questions an author might ask about the initial reference tool. This is also demonstrated by
showing an actual example of how one of the researchers used such questions and obtained the information needed to complete the description of the design pattern.

Also, the design pattern referenced in the walkthrough guide is also shown side-by-side, so that pattern authors can easily follow the entire process along.

Figure 5: The step-by-step walkthrough guide

4.2.7 List of Patterns in the Pattern catalog

We wanted our design pattern catalog to be easily accessible to all pattern authors and other visitors of the wiki. For this purpose, we created one page in the wiki which listed all the patterns currently present in our design pattern catalog.

Each entry in the list of patterns contained the following 3 components:

- A hyperlink to the actual design pattern
- A small icon indicating the completion status of the pattern. A✅ indicated that all required sections of the pattern description has been completed. A TODO indicated that some sections of the patterns were still a work-in-progress
• The Intent of the design pattern was also included in the entry, to provide a concise description of what problems the pattern is trying to address to visitors and pattern authors.

This list of patterns was generated programmatically, and any newly added pattern or change in completion status of any pattern would be automatically updated in the list.

Figure 6: Excerpt from the list of patterns in the wiki

4.3 Protocol for Design Pattern Submission and Review

After pattern authors provided their informed consent via an email reply, they were provided with the URL of the wiki and asked to create an account. They were also suggested to go through the supporting materials provided in the wiki such as the brief Primer on IFT, and a walkthrough of the Pattern writing process using an existing Design Pattern. Our research team was also in constant communication with the pattern authors via email to help them get familiarized with the process.

After reviewing the supporting materials, pattern authors began the process of drafting one or more design pattern of their own using authoring tools present in the wiki. The authors were also able to be post comments about the wiki material, as well as append
“Known Uses” (real-life instances of pattern implementation) to existing patterns, authored by other people.

The wiki was setup such that members of the study team received email notifications every time a modification was made to a Design Pattern in the wiki. After receiving notification that an author has created or modified a design pattern, the members of the study team logged into the wiki and reviewed the changes. After reviewing the changes, the study team privately communicated with the authors via email to give them suggestions on improving the pattern, as well as clarifying any inquiries they had about the pattern. Pattern authors followed up on this feedback by either explaining their reasoning or making further modifications. There were several iterations of this feedback-modification process until all members of the study team unanimously agreed that the pattern has fulfilled all the criteria present in the Design Pattern guideline. After reaching this conclusion, authors were asked if they wished to write additional patterns. If authors decided that their contribution was finished, they were provided compensation for their completed contribution. Authors were also free to withdraw from the study anytime, and were compensated for completed patterns and Known Uses prior to withdrawal.

Each design pattern author received $100 for each completed design pattern, up to 3. Patterns were considered to be complete if it was not a near duplicate of an existing pattern, all the sections were filled with relevant well-formed sentences, and any citation to an existing work was found to be authentic. This portion of the compensation did not require providing Known Uses to a pattern.

Design pattern authors were also paid $10 per each Known Use contributed, up to 20, with a maximum of up to 3 Known Uses per design pattern. A Known Use was considered complete if it had well-formed sentences that were on-topic, and it was accompanied with a reference that proved its authenticity.

**4.4 Design Patterns authored by our recruited authors**

The collection of design patterns from recruited authors lasted from April 2015 to January 2016. During this period, 9 different authors provided 16 design patterns. The
names of the design patterns, along with their corresponding intents (sometimes edited for succinctness) can be found in Table 2. The contributions of the recruited authors illuminated multiple insights absent from our earlier catalog.

Table 2: Design Patterns from our recruited authors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Processing</td>
<td>Give developers a high level description of source code, without having to navigate through the code.</td>
</tr>
<tr>
<td>Extract Method Refactoring</td>
<td>Restructure the topology by extracting statements that are highly related into a separate method and creating a new patch.</td>
</tr>
<tr>
<td>Fault Localization</td>
<td>Identify the sections of code that are responsible for an undesired behavior of software.</td>
</tr>
<tr>
<td>Heuristics-based Code Completion</td>
<td>For code completion, present the candidate functions grouped by their relatedness to the current coding context</td>
</tr>
<tr>
<td>Impact Location</td>
<td>Identify source code affected by the alteration of a different section of code.</td>
</tr>
<tr>
<td>Online Feedback Miner</td>
<td>Extract from forum discussions API features that have caused problems for developers</td>
</tr>
<tr>
<td>Patch Prevalence</td>
<td>Provide information foragers more prevalent patches so as to more quickly arrive at a potentially profitable patch.</td>
</tr>
<tr>
<td>Patch Profitability</td>
<td>Indicate how much value an entire information patch yields for fulfilling information-seeking goals</td>
</tr>
<tr>
<td>Path Search</td>
<td>Search a path in a topology, collapsing the topology to a list of prey containing cues matching the predator's information goal.</td>
</tr>
<tr>
<td>Recollection</td>
<td>Find a previously known class or method that is relevant to the task at hand.</td>
</tr>
<tr>
<td>Reduce Duplicate Information</td>
<td>Reduce the size of the topology by eliminating nodes with duplicate information.</td>
</tr>
</tbody>
</table>
**Rename Refactoring:** Rename methods to reflect information contained, highlighting aspects relevant to expected future foraging

**Shopping Cart:** Allowing developers to accumulate a list of patches for extra vetting

**Software Visualization:** Characterize domain elements, e.g. structural program elements, by visualizing metrics and properties

**Test Coverage:** Monitor coverage of a unit test suite to ease software maintenance and evolution

**Visualize Topology:** Reveal the structure of the topology, helping developers to move along relationships and choose patches to visit

For example, multiple design patterns (unlike the ones authored by our own research group) showed ways of reducing the cost of processing information patches. This was particularly apparent in design patterns summarizing information from text, thereby reducing the cost of navigating to disparate patches and/or mentally processing these patches. For instance, the *Documentation Processing* design pattern referred to tools that automatically parse and extract information from documentation into summative patches. The *Online Feedback Miner* pattern described tools, such as Haytack [57], that automatically digest online conversations to provide summative information to developers.

Other design patterns went beyond our own by describing how tools assist developers in making sense of large topologies, which our own design patterns had not extensively addressed. For example, the *Shopping Cart* cited Tracter, a tool for traceability analysts to collect patches in a topology so that they can subsequently view those patches and examine them in detail [37], and the *Visualize Topology* design pattern referred to tools that depicted the relationships among patches. One pattern, *Patch Prevalence*, described an approach for decreasing cost by increasing the density (“prevalence”) of high-value patches through, in essence, compressing or otherwise transforming the topology. It cited,
as an example, Code Bubbles, which enables the visual juxtaposition of high-value patches within a window offering low-cost between-patch navigations [10]. The Patchworks code editor supports a related approach with a similar effect [21].

Finally, compared to our own design patterns, history and temporal sequencing played a larger role in community-generated patterns. For example, the *Recollection* design pattern explained how tools can help developers find their way back to places that they have visited before. The *Rename Refactoring* and *Extract Method Refactoring* design patterns discussed tools that enable the developer to modify the topology in order to reduce the cost of future foraging activities by improving maintainability. Although our preliminary work had explained how tools can aid developers in finding information needed before performing refactoring tasks, we (unlike our community authors) had not made the connection between the act of refactoring and *future* cost of foraging.
5 Evaluation Part 1: Qualitative analyses of design pattern catalog

Our Design Patterns catalog consists of patterns contributed by our research group and the patterns contributed by the community authors in our study. Using a variety of qualitative analyses of our pattern catalog, we intend to answer the following Research Questions (RQs):

1. Is there an unequal problem domain coverage by tool builders/pattern authors?
2. How abstract are the patterns?
3. How widely used are the patterns in practice?
4. How are the Design Patterns inter-connected?
5. How “generalizable” are the patterns across different Software Engineering task types?
6. Are the community-generated patterns at least as “good” as those that we ourselves wrote?
7. What are the areas of design pattern description that pattern authors struggled with?

RQs 1 to 5 are about the entire design pattern catalog. RQ6 illuminates the “quality” of the community-generated patterns by comparing them with prior patterns authored by our research group. RQ7 makes certain observations from the experiences of design pattern authors.

5.1 Methodology

In this section, we briefly describe the methodology employed in answering each of these RQs.
5.1.1 RQ1: Is there an unequal problem domain coverage by tool builders/pattern authors?

To answer this research question, we looked at the design patterns in our catalog and categorized them according to the Information Foraging objective they tried to achieve.

To come up with the list of IFT objectives, we followed an open coding approach [3]. First, we considered all the constructs present in Information Foraging Theory, and how a tool might modify these constructs to assist in the various facets of information foraging. We also considered the temporal factor, as tools may try to achieve objectives that are immediate, or ones that are realized in future instances of foraging. The following table lists the IFT objectives and the corresponding rules we applied to each pattern to determine whether the pattern fulfilled this objective.

<table>
<thead>
<tr>
<th>IFT objective</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase the future value of information features</td>
<td>Some information features or cues have to be added to the current information features in the patch to make future processing of the patch easier and the changes are to be persisted.</td>
</tr>
<tr>
<td>Decrease the future cost of processing a patch</td>
<td>Irrelevant information features are removed from the patch. The changes made are persisted in some way, and they will be reused in the future.</td>
</tr>
<tr>
<td>Decrease future cost of navigation</td>
<td>New links are added to patches and they are persisted. The links will be reused in the future.</td>
</tr>
<tr>
<td>Allow better alignment of E[V] with actual V</td>
<td>Some information about the value of the information features in the patch are provided.</td>
</tr>
<tr>
<td>Allow better alignment of E(C) with actual C</td>
<td>Some information about how expensive it will be to process the patch is provided.</td>
</tr>
<tr>
<td>Decrease current cost of navigation</td>
<td>One-time only links are added to patches.</td>
</tr>
<tr>
<td>Decrease current cost of processing a patch</td>
<td>Irrelevant information features are removed from the patch. The changes made are ephemeral.</td>
</tr>
<tr>
<td>Increase current value of information features</td>
<td>Some information features or cues have to be added to the current information features in the patch to make processing</td>
</tr>
</tbody>
</table>
Locate the prey of interest for the predator | New links pointing to potential prey are provided.
---|---
Draw developer’s attention to certain information features | The stylistic appearance of the information features are changed.
Draw developer’s attention to certain cues | The stylistic appearance of the cues are changed.

One additional important thing to note is that we allowed a Design Pattern to fulfill multiple information foraging objectives.

To ensure reliability of our analysis, we followed the common inter-rater reliability (IRR) practices. Two researchers refined the IFT objective code set mentioned above, and then independently coded 20% of the Design Pattern catalog. A code set is generally considered as being reliable if coders independently achieve agreement of ≥ 80% on 20% of the data using the Jaccard index (the intersection of all applied codes over the union of all applied codes). We enforced this restriction on this code set and any code set used in subsequent qualitative analysis. For the IFT objectives code set, our inter-rater reliability was 80% on 20% of the data. After inter-rater reliability was achieved, one researcher completed coding of the remaining data. The result of this analysis can be found in the Results section of this document.

5.1.2 RQ2: How abstract are the patterns?

The level of abstraction across all design patterns is not the same. Some pattern descriptions contain extremely low-level implementation details, while others just describe a template solution, leaving the details vague/ to be implemented by the actual user of the pattern.

Taking these phenomena into consideration, we decided to classify the design patterns in our catalog in terms of levels of abstraction. For defining the abstraction levels, we used a modified version of the categorization used by Yskout et al.[1], adapted specifically for use in our design pattern catalog.
Yskout et al.’s abstraction level categorization consisted of 6 tiers of abstraction, ordered by decreasing levels of abstraction. However, their categorization was not directly applicable to our Design Pattern catalog, and so instead we just used a subset of their categories, and used our own ruleset appropriate and applicable to our own catalog.

Based on Yskout et al.’s categories, we defined our Patterns to belong to one of the two levels of abstraction, **Technique** and **Algorithm**, with Technique being at a higher level of abstraction than Algorithm.

We used a qualitative coding approach to classify Design Patterns as either being Technique or Algorithm.

If the solution described by the pattern had a well-defined context, and contained implementation details, then the Pattern was coded as being an Algorithm.

On the other hand, if the solution described by the pattern did not define its context, and implementation details were vague enough such that they differ from one context to another, then the Pattern was coded as being a Technique.

The standard inter-rater reliability practice was followed in this coding. Two researchers independently coded abstraction level of 20% of the patterns in the catalog, and achieved an agreement of 86%, thus achieving inter-rater reliability. The rest of patterns were then coded by one researcher.

### 5.1.3 RQ3: How widely used are the patterns in practice?

To answer this research question, we looked at the documented Known Uses of a pattern. We classified Known Uses into 3 categories:

- **Industrial Use**
- **Self-authored Research tool**
- **Research tool authored by others**

If from the official site of the tool, any of the following information about the tool was visible, then it was considered to belong to Industrial Use.
- The tool is available commercially
- The tool is under continuous development
- The tool has continuous releases
- The tools needs to be purchased for commercial usage
- The tool is open-source, but has option for paid support
- The tool has an active online community associated with it

If a feature of a mainstream IDE like Eclipse, Visual Studio, Intellij Idea was mentioned as a Known Use, the existence and description of the feature was confirmed from official documentation, and then was categorized as belonging to Industrial Use.

If from the official site of the tool, any of the following information about the tool was visible, then it was considered to belong to Research Tools. If the tool was authored by the pattern author, it was considered as a self-authored research tool. Otherwise, it was considered as belonging to the category of research tools authored by others.

- It is stated that the tool is part of an ongoing research project
- The tool is not available for public usage
- The tool appears to have been a one-off research project and is no longer under active development.

The standard inter-rater reliability practice was followed in this coding. Two researchers independently coded type of Known Use of 20% of the patterns in the catalog, and achieved an agreement of 94%, thus achieving inter-rater reliability. The rest of patterns were then coded by one researcher.

**5.1.4 RQ4: How are the Design Patterns inter-connected?**

As mentioned previously, we enforced that a design pattern description must contain a Related Patterns section. In this section, a pattern author described the ways this pattern was related with other design patterns in the design pattern catalog.

Thus, this section of pattern description provided us with the data to analyze the inter-connectivity between the design patterns in the catalog.
We followed a global search approach to at first identify the various possible relationships that could exist between a pair of design patterns. One researcher developed the initial set of relationship categories. Afterwards, two researchers cooperatively refined the initial code sets and developed a set of rules about applying them. Finally, the two researchers independently applied the code set on 20% of the data and achieved a reliability of 91.7%, thus achieving inter-rater reliability. The remainder of the data was coded by one researcher.

There were some general rules followed by researchers during this coding. These were:

- Unit of coding is pair of patterns
- Always use information from only the Related Patterns section.
- No inference is to be made by using information outside of Related Patterns.
- Burden of actually mentioning correct Related Patterns is to be left to Pattern Authors.

In addition to these general rules, the researchers also came with a set of rules per each Relationship category. The type of relations and the corresponding rules are described in Table 4.

Table 4: Type of Relations between Patterns

<table>
<thead>
<tr>
<th>Type of relations</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patterns A &amp; B have the same high-level purpose (but differ in implementation)</td>
<td>purpose must be mentioned explicitly, no inference should be made</td>
</tr>
<tr>
<td></td>
<td>a) It is mentioned in Related Patterns that both patterns fulfill the same information foraging objective</td>
</tr>
<tr>
<td></td>
<td>b) Both patterns help developer perform the same activity</td>
</tr>
</tbody>
</table>
| One pattern is a subpattern of the other (a refinement, or a specific case) | a) Mentioned explicitly in Related Patterns that one is a subpattern of the other  
b) mentioned explicitly that one pattern is a refinement of the other |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The patterns A &amp; B can be used concurrently with one another</td>
<td>Relation Patterns explicitly contains words like &quot;used together&quot;, &quot;used in combination with&quot;</td>
</tr>
</tbody>
</table>
| Pattern A can be used in a prior/subsequent step/phase before Pattern B | a) Pattern A fulfills some applicability precondition of Pattern B or vice versa. Note the ordering of the patterns.  
b) Usage of pattern A facilitates usage of Pattern B e.g. output produced by Pattern A can be used as input for Pattern B or vice versa. Note the ordering. |
| Patterns A & B use the same data as input | self-explanatory |
| None of the above | None of the above are applicable |

### 5.1.5 RQ5: How “generalizable” are the patterns across different Software Engineering task types?

To answer this research question, we looked at the documented Known Uses of a design pattern, associated the tool with the type of software engineering task it was assisting with and counted the number of software engineering task types the pattern was being applied to. The higher this count, the more “generalizable” this pattern was i.e. the pattern could be across several software engineering tasks.

To create the list of software engineering task types, we followed a grounded theory approach. We at first identified all the various software engineering task types that the patterns in our catalog have been applied to. One researcher developed the initial set of software engineering task types. Afterwards, several researchers collaborated to refine the initial set of tasks types into a more streamlined set of tasks and developed a set of rules
for applying the task types to design patterns. Finally, two researchers independently coded 20% of the data and achieved an agreement rate of 84.6%, thus achieving inter-rater reliability. The remainder of the data was coded by one researcher.

The list of software engineering task types along with a set of rules per each task are described in Table 5. We had 6 software engineering task types, but each task type had two subtasks. A set of rules developed corresponded to each subtask was developed, and if a Known Use met the criteria for a subtask, it was coded as belonging to the parent task of the subtask.

Table 5: Software Engineering Task Types

<table>
<thead>
<tr>
<th>Task commonly performed by Software Engineers</th>
<th>Subtask</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinating</td>
<td>Collaboration</td>
<td>Tool enables several developers, users, or other software engineering personnel (such as QA staff) to share information/communicate with each other; this does not include documentation, as that is given a separate category below</td>
</tr>
<tr>
<td>Notification</td>
<td>Tool updates developer about project status in some way (not just code or the project) -- Really notification of any sort</td>
<td></td>
</tr>
<tr>
<td>Understanding Code</td>
<td>Comprehension</td>
<td>Tool NOT PEOPLE adds extra information or information that is not easily available about source code; or the tool explicitly removes features with the specific purpose of making code more understandable</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>1. People NOT TOOLS can annotate artifact with comments about it</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Known use explicitly mentions “documentation”</td>
<td></td>
</tr>
<tr>
<td>Mapping to Code</td>
<td>Tool helps developer map functionality to source code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functionality being defined as something the program does while it is executing</td>
<td></td>
</tr>
<tr>
<td>Mapping Requirements</td>
<td>1. Tool helps developer map requirements/specification to code.</td>
<td></td>
</tr>
<tr>
<td>to Code</td>
<td>2. Traceability is explicitly mentioned in Known Uses</td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td>1. Known use explicitly mentions &quot;refactoring&quot;, &quot;reengineering&quot;</td>
<td></td>
</tr>
<tr>
<td>Code Reuse</td>
<td>1. Known use explicitly mentions &quot;Reuse&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Known use talks about taking existing code and using it again for a similar purpose (this includes libraries)</td>
<td></td>
</tr>
<tr>
<td>Debugging and Testing</td>
<td>1. Known Use specifically mentions &quot;testing&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Known use explicitly mentions &quot;debugging&quot;, stepping through code, tracing through code</td>
<td></td>
</tr>
</tbody>
</table>

**5.1.6 RQ6: Are the community-generated patterns at least as “good” as those that we ourselves wrote?**

In terms of pattern authors, pattern fall into three categories:

- Patterns originally authored in the TOSEM paper [16], referred from now on as TOSEM patterns
• Patterns authored by Researchers during the initial phases of this project, referred from now on as Local patterns

• Patterns authored by invited authors in the wiki, referred from now on as community-generated patterns

Thus far, we have evaluated patterns in terms of the IFT objectives addressed, and also in terms of abstraction, generalizability, type of Known Use. To determine if the patterns authored by invited authors are qualitatively as “good” as the ones we ourselves wrote, we compared the community-generated patterns with the TOSEM patterns in terms of these four criteria. The TOSEM patterns were chosen as a reference for determining “quality” because they had already been published in a peer-reviewed journal and thus can be said to have a certain baseline level of “quality”. If the community-generated patterns are found to have comparable values to the TOSEM patterns in all the criteria, then it can be said that the community-generated patterns are also as “good” as the TOSEM patterns.

5.1.7 RQ7: What are the areas of design pattern description that pattern authors struggled with?

Once a community author had contributed a design pattern, our research team reviewed the pattern and sent them feedback via email. The feedback followed a structured format. The research team reviewed the pattern one section at a time. If one section was decided to need further work, one paragraph in the feedback email to devoted to suggestions about the section. No feedback was given about sections no longer requiring any improvement. This structured format of the feedback allowed us to explicitly break down and measure exactly which areas of the design pattern description an author struggled with. If the feedback email contains a paragraph about a particular section, it was counted as the author having difficulty with that section. Each individual feedback email was counted as one unit. So, if an author was emailed twice about a certain section of a pattern description, it was counted twice.
5.2 Results from qualitative analysis of the Entire Design Pattern Catalog

5.2.1 RQ1: Is there an unequal problem domain coverage by tool builders/pattern authors?

To answer this research question, we looked at the design patterns in our catalog and categorized them according to the Information Foraging objective they tried to achieve. The detailed methodology of this categorization process can be found in the Methodology section.

After each pattern were categorized according to the IFT objectives they tried to address, we counted the instances of design patterns under each IFT objective category. The counts for each IFT objective reveal that currently there is an unequal problem domain coverage by tool builders.

![Figure 7: Patterns per IFT objective](chart.png)
From Figure 7, it can be seen that coverage of IFT objectives is not uniform. Several objectives are well-represented, while other are rather underrepresented.

The following IFT objectives are well-represented in the current Design Patterns catalog:

- Allow better alignment of \( E[V] \) with actual \( V \)
- Locate prey of interest for the predator
- Decrease future cost of navigation
- Decrease current cost of navigation
- Decrease current cost of processing a patch

On the other hand, the following IFT objectives are under-represented in the current Design Patterns catalog:

- Allow better alignment of \( E[C] \) with actual \( C \)
- Draw the attention of developer to certain information features
- Draw the attention of developer to certain cues
- Increase current value of information features
- Decrease future cost of processing a patch
- Increase future value of information features

This has several possible implications. One implication is that tools that address these under-represented objectives have not yet been documented by design patterns, and future iterations of this study/endeavor should have a stronger focus on documenting this patterns. Another implication may be that tool builders are not addressing these objectives enough, and future research should focus on improving the tool coverage of these objectives.
5.2.2 RQ2: How abstract are the patterns?

As mentioned in the Methodology section, we categorized the design patterns in terms of abstraction into two categories, Technique and Algorithm, with the former category being at a higher level of abstraction than the latter.

We used a qualitative coding approach to classify Design Patterns as either being Technique or Algorithm.

If the solution described by the pattern had a well-defined context, and contained implementation details, then the Pattern was coded as being an Algorithm.

On the other hand, if the solution described by the pattern did not define its context, and implementation details were vague enough such that they differ from one context to another, then the Pattern was coded as being a Technique.

We have currently categorized 34 design patterns based on their level of abstraction. Out of the 34, 18 are techniques and 16 are algorithms.

5.2.3 RQ3: How widely used are the patterns in public?

To answer this research question, we looked at the documented Known Uses of a pattern. We classified Known Uses into 3 categories:

- Industrial Use
- Self-authored Research tool
- Research tool authored by others

One of the restrictions we imposed was that each Known Use of a pattern must have an accompanying reference that authenticates it. To determine the type of Known Use, we visited its corresponding reference. If it was mentioned anywhere in the referenced document that the tool is available for commercial usage, we considered it to belong to Industrial Use. Otherwise, if the tool was found to be authored by pattern author, it was considered to be a self-authored research tool. All other Known Uses not falling into these two categories were considered as belonging to the category of research tools authored by others.
Our analysis revealed that out of all documented Known Uses, 65 of them are used for Research Purposes and 39 of them are already in industrial use.

Another important statistic was that we found that out of 34 patterns in our design pattern catalog, 19 of them already have at least 1 industrial use. This shows that about half of our patterns can be implemented as industrial solutions.

We also looked separately at the Known Uses documented by our study participants. 18 of the Known uses contributed by participants were industrial, 10 were self-authored research tools and 22 were research tools authored by others. This reinforces the replicability of the patterns contributed by our participants, as their authored patterns have also been used in both industry and other research projects.

5.2.4 RQ4: How are the Design Patterns inter-connected?

To answer this research question, we looked at the Related Patterns section of each pattern and identified pairs of patterns that were related. Then, we classified the relation between each pattern into the following categories:

- Patterns A and B have the same high-level purpose (but differ in implementation)
- One pattern is a subpattern of the other (a refinement, or a specific case)
- The patterns A and B can be used in conjunction with one another
- Pattern A can be used in a prior/subsequent step/phase before Pattern B
- Patterns A and B use the same data as input

The results from this analysis is presented in the form of a graph in Figure 8. The patterns are represented as nodes, and there are different types of edges between the nodes, with each edge type representing a particular category of relationship. Note that in certain cases, more than one type of relationships can exist between two patterns.
The results for each relation category is presented separately as well, to provide further insights into the nature of the relation categories.

**Patterns A and B have the same high-level purpose**

When we categorized two patterns as having the same high-level purpose, we also recorded the purpose itself. After the categorization was finished, we subsequently grouped together all patterns that fulfilled the same purpose. This was done to possibly improve to the usability of the design pattern catalog. If anyone wanted a design solution for a particular problem, this grouping according to purpose can provide the tool designer with the insight of which pattern to apply to achieve the desired purpose. This grouping of patterns according to purpose is presented in the following table.
Table 6: Patterns related by same high-level purpose (note: a pattern may have more than one high-level purpose)

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow developer to monitor for information</td>
<td>Dashboard, Notifier</td>
</tr>
<tr>
<td>find patches containing prey that are similar to the developer's query</td>
<td>Lexical Similarity, Semantic Clustering, Specification Matcher</td>
</tr>
<tr>
<td>find patches recently visited by predators</td>
<td>Personal Working Set, Past Aggregate Behavior</td>
</tr>
<tr>
<td>assist developers in automatically identifying the sections of code responsible for faulty behavior</td>
<td>Fault Localization, Regression Fault Localization</td>
</tr>
<tr>
<td>add cues to attract developer's attention or engender scent</td>
<td>Signpost, Cue Decoration</td>
</tr>
<tr>
<td>finds groups of code that are similar to each other in terms of functionality/behavior</td>
<td>Fault Localization, Semantic Clustering</td>
</tr>
<tr>
<td>create a single workspace containing collected information from different patches</td>
<td>Shopping Cart, Gather Together, Reduce Duplicate Information, Filtering</td>
</tr>
<tr>
<td>reduces the cost of processing information by removing the irrelevant information features</td>
<td>Reduce Duplicate Information, Shopping Cart, Filtering</td>
</tr>
<tr>
<td>locate prey in debugging tasks quickly and accurately</td>
<td>Path Search, Fault Localization</td>
</tr>
<tr>
<td>locate specific functionality within code</td>
<td>Path Search, Feature Tracing</td>
</tr>
</tbody>
</table>

**One pattern is a sub-pattern of the other**

In some cases, a particular design pattern was a refinement or a rather specific case of another, more abstract pattern. We isolated all patterns having this category of relationships to possibly improve the usability of the design pattern catalog. This category of related patterns can provide the tool designer with insight as to whether to apply the
parent pattern or the child pattern. Patterns along with their sub patterns are presented in Figure 9.

**Figure 9: Patterns having sub-patterns**

**The patterns A and B can be used in conjunction with one another**

In some cases, two patterns were found to be used concurrently in certain software engineering tools. We collected such patterns to possibly improve the usability of the design pattern catalog. These can possibly inspire the tool designer to think of innovative designs compared to viewing the patterns in isolation. The patterns which can used concurrently with one another are presented in Figure 10. Each pattern is presented as a node, and an edge between two nodes implies that the patterns can be used in conjunction.
Figure 10: Patterns which can be used concurrently

**Pattern A can be used in a prior/subsequent step/phase before Pattern B**

In certain scenarios, one pattern fulfilled some precondition of another pattern, or one pattern produced an output which was subsequently used by another pattern as input. Patterns which were found to follow such a sequential relationship are presented in the following figure.
When we categorized two patterns using the same data as input, we also recorded the data they were using. We subsequently grouped together all patterns using the same data as input. This was done to possibly improve the usability of the design pattern catalog. If anyone wanted to design a new feature using a particular type of data, this grouping according to same data can provide the tool designer with the insight of which pattern to implement using the data.
5.2.5 RQ5: How “generalizable” are the patterns across different Software Engineering task types?

To answer this research question, we looked at the documented Known Uses of a design pattern, associated the tool with the type of software engineering task it was assisting with and counted the number of software engineering task types the pattern was being applied to. The higher this count, the more “generalizable” this pattern was i.e. the pattern could be across several software engineering tasks.

We have currently categorized 34 design patterns based on this measure of “generalizability”. Out of the 34 patterns, 12 of the patterns could be applied more than 1 software engineering task types. As an example of a pattern that can be applied to more than 1 software engineering task, we can consider the Bookmark pattern. Bookmarks are used in **Debugging and Testing** in the form of breakpoints, while IDEs like Eclipse, Visual Studio allow creation of bookmarks in code, thus helping in **Coordination**.

5.3 RQ6: Are the community-generated patterns at least as “good” as those that we ourselves wrote?

To determine if the patterns authored by participants are qualitatively as “good” as the ones we ourselves wrote, we compared the community-generated patterns with the TOSEM patterns in terms of these four criteria. The TOSEM patterns were chosen as a reference for determining “quality” because they had already been published in a peer-reviewed journal and thus can be said to have a certain baseline level of “quality”. If the community-generated patterns are found to have comparable values to the TOSEM patterns in all the criteria, then it can be said that the community-generated patterns are also as “good” as the TOSEM patterns.

5.3.1 Criteria: IFT objectives addressed

The following chart provides the patterns per IFT objective for the TOSEM patterns.
Figure 13: Patterns per IFT objective: TOSEM and community-generated patterns

As can be seen from Figure 13, IFT objectives addressed most frequently by TOSEM patterns are:

- Locate the prey of interest for the predator
- Allow better alignment of $E[V]$ with actual $V$
- Decrease future cost of navigation
- Decrease current cost of navigation

As can be seen from Figure 13, IFT objectives most frequently addressed by community-generated patterns are:

- Allow better alignment of $E[V]$ with actual $V$
- Decrease current cost of processing a patch
- Decrease current cost of navigation
- Locate the prey of interest for the predator
Three of the most frequent categories (Allow better alignment of E[V] with actual V, Locate prey of interest for the predator and Decrease current cost of navigation) are the same for both TOSEM patterns and community-generated patterns. However, the other most frequently addressed category by community-generated patterns, Decrease current of processing a patch, was one of the infrequently addressed categories by the TOSEM patterns. It can also be seen from the chart that community-generated patterns has addressed two objectives: Increase current value of information features and Allow better alignment of E[C] and actual C which were not addressed by any of the TOSEM patterns.

This suggested that the participants possibly did a better job of covering the IFT objectives. To confirm this, we counted the number of IFT objectives that had at least a certain number of TOSEM patterns and also the number of IFT objectives that had at least a certain number of community-generated patterns. The resulting data was plotted on a chart.

![Image of chart](image.png)

**Figure 14: Objective coverage level comparison**

As can be seen from Figure 14, it can be clearly seen that for every level of coverage (number of patterns), our participants did an equal or better of covering the IFT objective.
The preliminary TOSEM catalog only addressed 8 objectives, while the community-generated patterns covered 10. Moreover, for any given level of coverage (expressed as a number of design patterns addressing each objective), the community-generated patterns met or outperformed the TOSEM patterns (Figure 14). Therefore, our participants covered the IFT objectives quite well relative to the TOSEM patterns.

This provides encouraging signs for future passes of the pattern collection process via a community wiki. Future iterations may result in increased coverage of other IFT objectives having low coverage.

5.3.2 Criteria: Level of Abstraction

Prior qualitative evaluation of design patterns have often used abstraction level as a metric for pattern quality e.g. for evaluation of security patterns[56] and multi-agent systems[24]. Pattern abstraction is valuable because it may indirectly support generalizability (below), although we suspect that the loss of concreteness might trade off in terms of the utility of the pattern for training tool designers, so it is important to assess a pattern catalog’s overall level of abstraction. Out of 12 TOSEM patterns, 9 were found to be Techniques, while 3 were found to be Algorithms. Whereas, out of 16 community-generated patterns, 7 turned out to be Techniques while 9 were found to be Algorithms. This means that the community-generated patterns were generally less abstract and more specific compared to the TOSEM patterns.

A possible explanation for this maybe the fact that the TOSEM patterns were uncovered by the researchers after a thorough and detailed review of existing software engineering research literature. Because of this they were able to generalize various aspects of the tools at a more abstract level. Whereas, participants may have initially used one research tool (possibly self-authored) as a reference and reverse-engineered a design pattern from it, making the pattern more closely tied to implementation details.

This result can actually be interpreted in several possible ways. While prior works on evaluation of design patterns have used abstraction as a quality metric, it can be argued that the abstraction level of a pattern might not be the best indicator of the “quality” of a pattern. For example, in case of IFT-based design patterns, a more abstract pattern can
perhaps be applied to a variety of software engineering tasks. However, from the perspective of a programming tool designer, a design pattern with a low-level abstraction might actually be a lot more useful. The presence of low-level implementation details would make it easier for the tool designer to actually implement the design pattern in a programming tool.

So, from one perspective, the community-generated patterns being less abstract than the TOSEM patterns makes them easier to implement and thus more “useful” from the perspective of a tool designer. From another perspective, the more abstract TOSEM patterns are applicable across multiple domains of software development. This exposes the limitation of using abstraction as a quality metric for IFT-based design pattern catalog, as patterns at both levels of abstraction are useful in different ways.

5.3.3 Criteria: Generalizability

Out of the 12 TOSEM patterns, 6 of the patterns could be applied to more than 1 type of software engineering task, thus being “generalizable”. Thus about half of the TOSEM patterns were generalizable.

Out of the 16 community-generated patterns, 4 of the patterns could be applied to more than 1 type of software engineering task. Thus only about a quarter of the community-generated patterns were generalizable.

This means the community-generated patterns were generally more tied to one specific software engineering task type.

A possible explanation for this is that participants may have initially used one research tool (possibly self-authored) as a reference and reverse-engineered a design pattern from it, making the pattern more closely tied to software engineering task the initial reference tool was used for.

This result reinforces the result from comparing the TOSEM and community-generated patterns in terms of level of abstraction. The TOSEM patterns were more abstract and hence more applicable to several software engineering tasks, whereas the community-generated patterns were more specific and thus were tied to one specific type of software engineering activity. Both approaches have their own complementary strengths. The
complementary nature of both levels of generalizability indicates a potential limitation of generalizability as “quality” metric for IFT-based design patterns.

5.3.4 Criteria: Type of Known Uses

Out of the 38 documented Known Uses in the TOSEM patterns, 13 of those were industrial and 25 of those were research tools authored by others.

Out of 50 documented Known Uses in the community-generated patterns, 18 of those were industrial and a total of 32 of them (10 self-authored and 22 authored by others) were research tools.

![Bar chart showing average numbers of Known Uses cited per design pattern categorized by type ofKnown Use's tool.]

Figure 15. Average numbers of Known Uses cited per design pattern (categorized according to whether each Known Use’s tool was industrial, was created by the pattern author’s research group, or was created by another research group)

As shown in Figure 15, the TOSEM catalog average 3.17 Known Uses per design pattern, slightly exceeding the 3.13 per design pattern that our community authors provided. However, the TOSEM patterns provided slightly fewer industrial use cases per design pattern, at 1.08 compared to 1.38 for community-generated design patterns. The authors of the TOSEM patterns did not provide any Known Uses regarding their own prior tools; in contrast approximately 1/3rd of research tools cited by community authors were created by their own groups. Overall, these results suggest that community-generated design patterns compared well to our own in terms of their evidence for use in practice.
5.4 RQ7: What are the areas of design pattern description that participants struggled with?

To identify the areas of design pattern description that our participants struggled with, we looked at the list of feedback emails per participant-contributed pattern. The feedback emails had a well-defined structure: if a section of a design pattern required feedback, the feedback email contained a paragraph containing suggestions for that section. If a section was considered finalized, the feedback email did not contain any paragraph for that section. If the feedback email contained a paragraph about a particular section, it was counted as the participant having difficulty with that section. Thus, if the feedback count for a particular section was higher compared to the count for other sections, it can be interpreted as participants having greater difficulty in describing that particular section of a pattern. The chart in Figure 16 presents the results of this analysis.

![Feedback count per section chart]

Figure 16: Feedback count per Design Pattern section

As can be seen from Figure 16, the two sections participants struggled with are Connection to IFT and Related Patterns.
Regarding participants requiring more feedback for **Connection to IFT** sections, it must be mentioned that this was not about participants failing to express core concepts of programming tools in terms of IFT. In fact, the majority of the participants were able to map the core concepts of software engineering tool design to IFT. Rather, participants demonstrated some difficulty connecting/mapping some of the more peripheral elements involved in their pattern description to IFT concepts. The feedback for such cases consisted of the researchers identifying such potential IFT connection opportunities and alerting the pattern writer to these. Some examples of such feedback is given below. Just like in the actual feedback email, the pattern sections and IFT constructs/concepts are bolded.

“**The Connection to IFT section** does a good job of identifying **prey** and **predator**. However, from the **Description** and **Known Uses**, I also see additional details that can be added here. In IFT terms, the tool is reducing the **cost of navigation** by providing **links** to potential **prey**, and the associated **cues** (the text of the links) are allowing **predator** to **estimate the value**, thus helping him make the decision of where to navigate to. Do you agree with this?”

“In the **Description section**, it is said “the tool can automatically find these clues, parse the comments, and create the documentation pages”. In IFT terms, it looks like the **prey** (the documentation) are extracted from their normal locations in source code to a completely new **information patch** (the documentation pages) which reduces the **cost** of finding documentation. This is **enrichment** of the topology, and should probably be mentioned in the **Connection to IFT** section.”

“Although it is mentioned in other sections, the **Connection to IFT** section can also mention that the profitability is presented as a **cue** to the **patch** itself.”
“The Connection to IFT section does a good job of identifying prey and predator. It also mentions that the pattern engenders information scent. However, I think it should also be mentioned that the metric indicating coverage is the one information feature that acts the cue.”

In some cases, the **Description** section did not contain enough details. However, if the researchers estimated that certain elements would end up being included in the **Description** section and would subsequently end up affecting the **Connection to IFT** section as well, the researchers provided this feedback to the pattern writer as well.

“The **Description** sections lacks detail about how the mined/extracted features are presented to the developer. Is it a ranked list of extracted features, with links to the actual post in the forum?

How the results are presented may affect **Connection to IFT** section, because if the result contains hyperlinks to actual post it is reducing **cost of navigation** by proving a **link** having lower **cost**. The ranking values may also act as **cues** engendering **information scent**.”

The other section in which participants required frequent feedback was **Related Patterns**. Participants were often able to identify at least one pattern that related to the current pattern. However, sometimes they did not provide enough detail about the exact nature of the relation between the patterns. Researchers requested participants to elaborate on this.

“**Gather Together** pattern is mentioned, but exactly how **Gather Together** is related and how it is different from this one is not mentioned.”

“In **Related Patterns**, it would be helpful if the difference between **Test Coverage** and **Regression Fault Localization** is made explicit.”
“Two related patterns are mentioned, but exactly how these patterns are related and how Software Visualization is different from them is not mentioned.”

In other cases, researchers also identified additional patterns which could be potentially related to the pattern and alerted the pattern writer to this.

“Also, it seems that the Filtering Pattern is also somewhat similar to this one as Filtering also helps filter out irrelevant results. So the similarity and difference between this one and Filtering can also be discussed.”

“Since this pattern uses NLP techniques so much, I feel this is a subpattern of Lexical Similarity for a very specific use-case. Even if it is not directly a subpattern, it certainly prominently uses NLP techniques, and thus its Related Patterns section should mention Lexical Similarity for this reason.”

“In Related Patterns section, it sounds like it has some similarity with Feature Tracing. Here the tool automatically finds cues to fault locations from source code and runtime output, whereas Feature Tracing lets developer specify how to make the feature/fault execute, and then executes the feature/fault and reports the cues from execution trace.”

A potential explanation for this is that participants did not read through and understand all the patterns in the design pattern catalog. Rather, they just read about patterns which sounded the most similar to their contributed pattern based on the pattern intent present in the “List of Patterns” page (example of the page can be seen in Figure 6). On the other hand, researchers conducting the study continuously reviewed all the patterns in the catalog, and hence were able to make connections between patterns participants were unable to make.
6 Evaluation Part 2: Assessing the usefulness of the Design Pattern catalog

Prior research aimed at collecting design pattern catalogs primarily focused on qualitative analyses of quality, but we wanted to assess the practical usefulness of our IFT-based catalog. Therefore, we conducted additional empirical studies that involved collecting feedback from two potential user groups: professional tool designers and graduate students in computer science.

6.1 Methodology

6.1.1 Obtaining Feedback from Professional Tool Designers

Once the process of collecting design patterns from software engineering researchers was finished, we began the process of evaluating the usefulness of the design pattern catalog for actual programming tool designers.

The first phase of evaluation involved one of the researchers presenting the catalog as part of a talk at Google Inc., demonstrating the usage of the pattern catalog to solve a programming tool design problem. The content of this talk consisted of a brief introduction to the concepts of IFT, a brief introduction to IFT-based design patterns, an example programming tool design problem and finally a step-by-step demonstration of applying the design pattern catalog to arrive at a solution to the design problem.

After the demonstration, the attendees of the talk were asked to evaluate the usefulness of the design pattern catalog by completing a questionnaire.

The questionnaire could broadly be divided into 3 categories:

- Demographic information – this section collected demographic data such as participant’s primary job role, level of experience and their level of contribution to design of programming tools
- Open ended feedback – This section had the following open ended questions about design patterns:
• Likert scale Survey questions – This section contained several statements about the usefulness of the design patterns, with participants’ options for agreeing with these statements being a 5 point Likert scale of ranging from “Strongly Disagree” to “Strongly Agree”. The questions were:

  o “I understand the purpose of the design pattern catalog”
  o “I understand the basics of Information Foraging Theory”
  o “I understand how to find useful design patterns in the catalog”
  o “I am comfortable using the catalog to design programming tool features”
  o “The catalog makes it easier for me to design programming tool features”
  o “The catalog helps me think of designs that I probably would not think of my own”
  o “As a professional tool designer (or I were), I’d probably use the catalog”

We later used the data from the completed questionnaire to determine what the participants thought about the usefulness of the design pattern catalog.
6.1.2 Measuring the utility of the patterns for training graduate students

Another potential application of the design pattern catalog is to facilitate the training of novice programming tool designers. Novice programming tool designers can look at the catalog and educate themselves about the programming tool landscape, obtain further insights into what makes programming tools more effective, as well as gain knowledge about potential tool designs previously unknown to them.

We are considering multiple forms of dissemination for our design pattern catalog, including online media as well as a traditional textbook.

To evaluate the effectiveness of the online versus printed formats of the design pattern catalog for training novice tool designers, we conducted a study among the computer science graduate students at Oregon State University.

We recruited 24 graduate computer science students studying at Oregon State University for this purpose. A brief description of what the study would entail, along the enrollment form, was disseminated among the students via Oregon State University’s mailing list and professors teaching relevant courses. In order to ensure that participating students have the proper background required for the study, the enrollment form required students to confirm that they had some level of professional experience as a software engineer. Participants could sign up for one of the two workshops being offered. One workshop was for the treatment condition and the other was for the control condition, but participants were not made aware of this. Participants were also paid a remuneration of $100 for attending the workshop.

At the beginning of each workshop, we gave participants a brief overview of what the workshop would entail, distributed consent forms among them, answered any questions they might have had and finally obtained signatures from them.

The workshop was divided into 3 main phases. In the first phase, we gave a lecture introducing the training materials for the workshop. For the treatment condition, the training materials consisted of the design patterns wiki. For the control condition, the training materials consisted of a compilation of the tools mentioned in the Known Uses
section of the design patterns in the wiki. Each tool had a text-based description, accompanied by a screenshot if available. In case of the tool being a research-based one, we also included a citation of the corresponding research paper, as present in the wiki. Tools were presented in the same order that they appeared in the design pattern catalog.

In each case, our lecture was designed to motivate the need for IFT and associated training materials.

It should be noted that neither of the two conditions was a perfect control in the sense of replicating OSU’s graduate software engineering course, which has students read research papers. Rather, our study compared the effectiveness of (a) the online design pattern catalog with b) a printed catalog of screenshots for Known Uses and accompanying text.

To illustrate the use of these materials to design a software engineering tool, we presented a sample software engineering tool design problem, and used the training materials to design a tool aimed at addressing the problem. We expressed the proposed design in the form of a paper prototype. We also had participants interact with the prototypes as it was an actual computer screen, keeping consistency with standard paper prototyping methods.

The above procedure was repeated with a second design problem. However, this time, we interacted with the participants to allow them to contribute to the design of the tool. This concluded the tutorial phase of the workshop.

In the second phase, we divided the participants into pairs. Each person in a pair was given a separate design problem (Task A for one participant and Task B for the other participant) and asked to produce a paper prototype of a tool. Participants were told to use our provided training materials to inform their design.

Once a pair of participants completed their design, they were asked to exchange their produced paper prototypes and interact with them.

In the third phase, we distributed a written questionnaire, asking participants about their background, agreement with a set of statements regarding the usefulness of the
training materials presented, and agreement with a set of statements about the quality of the paper prototypes participants interacted with.

The questionnaire could broadly be divided into the following three categories:

- **Demographic information** – this section collected demographic data such as participant’s level of professional software engineering experience, the degree being pursued by the participant and in which year of graduate school they were in.

- **Likert scale Survey questions about the training materials** – This section contained several statements about the usefulness of the design patterns, with participants’ options for agreeing with these statements being a 5 point Likert scale of ranging from “Strongly Disagree” to “Strongly Agree”. The statements for the treatment group of the workshop were:
  
  - I understand the purpose of the purpose of the design pattern catalog
  - I understand the basics of Information Foraging Theory
  - I understand how to find useful design patterns in the catalog
  - I am comfortable using the catalog to design programming tool features
  - The catalog makes it easier for me to design programming tool features
  - The catalog helps me think of designs that I probably would think not think of on my own
  - If I was a professional tool designer, I would probably use the catalog
  - I plan to use the design pattern catalog in my courses and/or research at OSU

- For the control group, the same set of statements were used, but the phrase “design pattern catalog” was replaced with “catalog of existing programming tools”.

- **Likert Scale Survey questions about the paper prototypes produced** – This section contained several statements about the quality of the paper prototype
that a participant interacted with in the previous section of the workshop. The participants’ options for agreeing with these statements being a 5 point Likert scale of ranging from “Strongly Disagree” to “Strongly Agree”. The statements were:

- I understood the purpose of the feature(s) that the prototype(s) depicted
- I understood how to interact with the feature(s) that the prototype(s) depicted
- The feature(s) probably would be useful, if actually implemented.

- The questionnaire for the treatment group also had a couple of open-ended questions:
  - What aspects of the design patterns seem most useful?
  - What aspects of the design patterns seem least useful?

6.2 Results

6.2.1 How useful do tool designers find the design pattern catalog to be?

We briefly summarize the results from analyzing participants’ response in the questionnaire given out after the talk on the pattern at Google, Inc. A total of 11 participants answered the provided questionnaire. Out of these 11, 7 were programmers/developers/software engineers, 1 was a researcher, 1 was a Technical Writer, 1 was a Program/Product Manager and 1 declined to answer. 10 out of 11 participants had a job experience of more than an internship, with 7 of them having a job experience of 3 years or more. Also, 8 out of 11 participants have had involvement in development of at least 1 programming tool.

So from the demographic data, it can be seen that a majority of the participants at the talk were programmers of moderate experience. A majority of them of also had some level of experience in development of a programming tool.
The summary of participants’ response from the Likert scale survey about the patterns’ usefulness is given below.

1. Everyone out of the 11 participants agreed that they understood the purpose of the catalog.
2. 10 out of 11 participants agreed that they understood the basics of IFT. 1 participant remained neutral. No one disagreed.
3. Everyone out of the 11 participants agreed that they understood how to find useful patterns in the catalog.
4. 9 out of 11 participants agreed that they would be comfortable using the catalog to design programming tool features. 1 participant remained neutral, nobody disagreed.
5. 7 out of 11 participants agreed that the catalog made it easier for them to design programming tool features. 3 participants remained neutral on this. 1 participant declined to answer.
6. 9 out of 11 participants agreed that the catalog helps them think of designs they would not have thought of on their own. 1 participant remained neutral on this. 1 declined to answer.
7. 9 out of 11 participants agreed that they would use the catalog when designing tools. 2 participants remained neutral.

So, from the above results it can be seen that the overwhelming majority agreed or strongly agreed with every statement about the catalog’s utility. It was possible to combine the 7 answers into a scale with Cronbach’s alpha of 0.96. The average score for this was 4.28 on a scale of 1-5. This provides evidence that the design pattern catalog provided guidance considered highly useful by professional tool designers who were presented with it.

Participants were also asked to give open-ended feedback on several aspects of the catalog, such as most and least useful aspects of the catalog. Participants often stated that abstracting and gathering existing tool design concepts into patterns was the most useful
aspect of the catalog. Some of the actual responses of the participants regarding this are quoted below.

“Being a shared knowledge and providing vocabulary”

“crystallizing concepts related to designing programming tools into a concrete list”

“production examples, past results”

“exemplar, idea generation”

The most frequent suggestion about potential improvements to the pattern catalog was a) to provide some of visual examples of the pattern being applied and b) to tag patterns with the type of tasks they can be applied to. Some of the actual responses of the participants regarding this are quoted below.

“The patterns themselves are very text heavy. Screenshots of examples would make it easier to quickly grasp what particular pattern is about”

“I would like to see some visual examples of those patterns to better understand how these are applied”

“domain tags e.g. patterns that apply in rich graphical media like web views vs. in-code or API design affordances”

“Easier ways to find out relevant design patterns for task”
6.2.2 How useful is the pattern catalog for training graduate students?

In this section we analyze the results from the study conducted with graduate students at Oregon State University, in which participants were trained to come with design of programming tools using Condition 1) The Digital Wiki and Condition 2) Printed Known Uses. After the training, we divided them into pairs, gave each person in a pair their own design task (Task A for one person in the pair, and Task B for the other person). The participants were asked to come up with a paper prototype using the given training materials. Once finished, the other person in the pair interacted with paper prototype. Finally, participants completed a questionnaire containing statements about the utility of the training materials and the produced paper prototypes.

A total of 24 students participated in the study. Of the 24 participants, 17 reported having professional experience in software engineering beyond just an internship; 12 were Master’s students, and 12 were PhD students; 17 of 24 were in their second year or later.

All answers to the questionnaire questions asking about agreement regarding the utility of the training materials were into a scale having a Cronbach’s alpha of 0.80 on the participants’ data, indicating sufficient reliability for retaining the scale.

We also attempted to synthesize a second scale from the three additional items in the questionnaire about the usability of the paper prototypes, but the resulting Cronbach’s alpha score was only 0.54, so the results for these three questions are reported separately. Due to small, non-normal distribution of answers, we tested between condition#1 and condition#2 using non-parametric two-tail Kruskal Wallis test. We similarly tested between Task A and Task B.

The results from the analysis about agreement regarding catalog’s usefulness can be found in the two rightmost columns of Table 7. The results from the analysis about the paper prototypes produced by participants can be found in Table 8.

At the level of p < 0.05, we did not find any significant differences between Task A and Task B on any of the measures, nor between Condition #1 and Condition #2. The
overall average scale score of 4.13 was comparable to that given by the Google staff, and the average scores on the three prototype evaluation questions were in the range 4.17-4.79, suggesting the comparable value of both online and printed materials for training students in tool design. In addition, medians were at the level of “agree” or “strongly agree” regarding statements of the prototypes’ usability (Table 8).

The questionnaire also collected qualitative feedback from the participants. These confirmed the information we had obtained from the Google study regarding the relative strengths of the two conditions. As with the Google Staff, students emphasized the value of and need for, visual imagery depicting the application of the design patterns to actual programming tools. For example, one participant wrote “I wished I could see some visual examples and Figures in the wiki” and another participant noted that the wiki had “no/few illustrated examples of design.” Yet like the Google staff, they valued the organization of examples around patterns on the wiki. One participant wrote “The classification of the different patterns and their purpose were the most useful”, and another wrote “The design patterns gave me a global view of necessary tool designing requirements.”

Based on these results, we intend to synthesize the best of both teaching methods by integrating screenshots into the digital wiki, which we will then likely convert into a printed book for the purpose of dissemination and broader impact.
Table 7: Summary of Agreement with statements (1=Strongly Disagree through 5=Strongly Agree); Google Column shows agreement by industrial tool designers (Section 6.2.1) and the rightmost two columns show agreement by graduate students (Section 6.2.2)

<table>
<thead>
<tr>
<th>Statements about the catalog’s usefulness (We combined responses to these statements into a scale)</th>
<th>Google (n=11): Digital wiki</th>
<th>Student Workshop Condition 1 (n=12): Digital wiki</th>
<th>Student Workshop Condition 2 (n=12): Printed Known Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>• I understand the purpose of the catalog</td>
<td>Overall, 100% of participants agreed (scale score &gt;= 4.0)</td>
<td>Overall, 83% of participants agreed (scale score &gt;= 4.0)</td>
<td>Overall, 100% of participants agreed (scale score &gt;= 4.0)</td>
</tr>
<tr>
<td>• I understand the basics of Information Foraging Theory</td>
<td>Median: 4.27</td>
<td>Median: 4.00</td>
<td>Median: 4.25</td>
</tr>
<tr>
<td>• I understand how to find useful patterns in the catalog</td>
<td>Average: 4.28</td>
<td>Average: 3.90</td>
<td>Average: 4.23</td>
</tr>
<tr>
<td>• I am comfortable using the catalog to design programming tool features</td>
<td>s.d.: 0.23</td>
<td>s.d.: 0.51</td>
<td>s.d.: 0.60</td>
</tr>
<tr>
<td>• The catalog makes it easier for me to design programming tool features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The catalog helps me think of designs I probably would not think of on my own</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• As a professional tool designer (or if I were), I'd probably use the catalog or I plan to use the design pattern catalog in my courses and/or research at OSU</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8: Statements about paper prototyped features between Condition 1 (Digital Wiki) and Condition 2 (Printed Known Use)

<table>
<thead>
<tr>
<th>Statement</th>
<th>100% agreed; median 5.0</th>
<th>100% agreed; median 5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>I understood the purpose of the feature(s) that the prototype(s) depicted</td>
<td>100% agreed; median 5.0</td>
<td>100% agreed; median 5.0</td>
</tr>
<tr>
<td>I understood how to interact with the feature(s) that the prototype(s) depicted</td>
<td>100% agreed; median 4.0</td>
<td>92% agreed; median 4.5</td>
</tr>
<tr>
<td>The feature(s) probably would be useful, if actually implemented</td>
<td>83% agreed; median 5.0</td>
<td>75% agreed; median 4.0</td>
</tr>
</tbody>
</table>
7. Limitations and Threats to validity

7.1 Pattern Collection

A limitation for our pattern collection process is that we selected our pattern authors based on research they had published in well-known software engineering conferences. When trying to author design patterns, it is possible the authors drew on the knowledge from their own specific domain of expertise. Thus, the lack of coverage of specific topics is possibly related to us not having recruited authors specializing in those topics, and not necessarily due to those topics not being worked by the research community.

Qualitative analysis also revealed the community-generated patterns to be of lower generalizability and abstraction compared to the prior TOSEM patterns. In the future, a literature review or some other post-processing step might be taken to ensure generalizability.

7.2 Qualitative Analysis and Practical Evaluation

In our qualitative analysis we used two metrics, Abstraction level and Generalizability as part of a set of metrics to judge pattern “quality” due to their usage in prior design pattern catalog evaluations. However, for IFT-based design patterns, arguments can be made about the usefulness and strength for both concrete, specific patterns and more abstract, generalizable patterns. Thus, these two metrics are quite limited for indicating pattern “quality”.

A threat to the internal validity of our practical evaluation is that we did not ask Google staff or student participants to implement tools using design patterns: our measurement of opinions about utility might not match measurements of utility in practice. To limit this threat, we asked student participants to create paper prototypes, which we also evaluated. Future work could investigate the extent to which tool designers benefit from our design pattern catalog when turning designs into working code.

A third threat, to external validity, is the potential that Google employees’ opinions about the design pattern catalog’s utility might not match those of other tool designers, and
the experiences of Oregon State University graduate students might not match those of other novice tool designers; research could address this threat through a replicated study.
8. Conclusion

We have presented a community-generated design pattern catalog that expands upon our initial collection of IFT-based design patterns for programming tools. The new contributions expanded the catalog in innovative directions, such as through the inclusion of design patterns showing how to reduce the cost of understanding code, as well as showing how to make sense of large topologies (through features for visualization or visual organization).

We have also presented a qualitative analysis of the entire design pattern catalog. It was found that there is an unequal problem domain coverage in the catalog thus far. Very few patterns address aligning expected cost with actual cost. Also, very few patterns draw attention to information features. Relatively few patterns strive to assist developers in lowering future cost of processing or increasing future value of information features as well. It was also found that overall, there was roughly an equal distribution of patterns in terms of abstraction level (half were at a higher level of abstraction, while the rest were at a lower level). It was also found that our design patterns catalog had a higher fraction of research-based tools implementing the patterns, compared to industrial tools. Analyzing the entire catalog in terms of “generalizability” revealed that roughly one-third of the patterns were generalizable i.e. could be applied across several software engineering tasks.

Our catalog-wide analysis also found the existence of several specific type of relationships existing between the design patterns. We later added the results from this analysis to our digital Wiki to potentially improve the usefulness of the pattern catalog for future users.

Finally, we also performed a qualitative analysis of our participants’ difficulties during the pattern authoring process. This revealed that participants had the most difficulty with adding “Connection to IFT” and “Related Patterns” to the design patterns.

We also evaluated the quality of the community-generated patterns relative to our preliminary collection of the 12 patterns published previously in TOSEM [16]. Overall, we found that the new design patterns from the research community compared well to our own
in coverage and evidence of use. They were also more specific and concrete, although less
generalizable than our own. Industrial tool designers gave high ratings to the utility of the
design pattern catalog, as did computer science graduate students who used training
material derived from IFT in digital and printed form.

These results: (1) further establish the relevance of IFT for tool design; (2) demonstrate the feasibility of connecting software engineering theory to practice via a
community-based process; (3) demonstrate that there is an unequal coverage of problem
domains by programming tool designers; (4) identify potential areas of difficulty for future
iterations of collecting patterns via community-based process; (5) expand our preliminary
collection on multiple dimensions of quality; (6) provide guidance that professional tool
designers considered highly useful; and (7) show the viability of disseminating training
materials to students digitally and in print.

Most importantly, however, our work illustrates a process connecting theory with
the day-to-day needs of tool designers. In doing so, it lights a path towards an ambitious
goal: to establish a means of bringing a scientific theory of human behavior to the practice
of building software tools for developers.
Bibliography


and comprehensible design method. *IEEE/IFIP Conference on Software Architecture* (WICSA), 157-166.
Appendices