FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Department Informatik

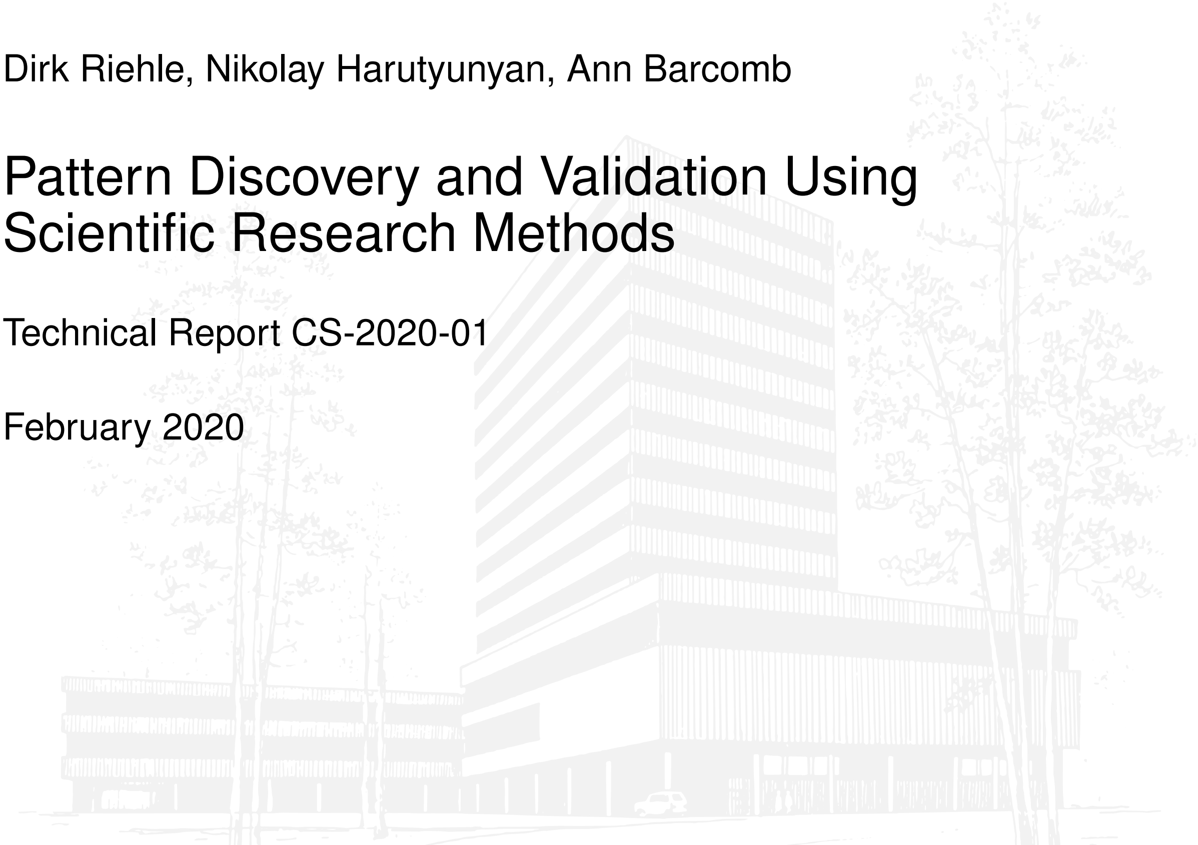
Technical Reports / ISSN 2191-5008

Dirk Riehle, Nikolay Harutyunyan, Ann Barcomb

Pattern Discovery and Validation Using Scientific Research Methods

Technical Report CS-2020-01

February 2020



Please cite as:
Dirk Riehle, Nikolay Harutyunyan, Ann Barcomb, “Pattern Discovery and Validation Using Scientific Research Methods,” Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2020-01, February 2020.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik

Martensstr. 3 · 91058 Erlangen · Germany

www.cs.fau.de

Pattern Discovery and Validation Using Scientific Research Methods

Dirk Riehle[1], Nikolay Harutyunyan, Ann Barcomb

Friedrich-Alexander-University Erlangen-Nürnberg

Abstract

Pattern discovery, the process of discovering previously unrecognized patterns, is usually performed as an ad-hoc process with little resulting certainty in the quality of the proposed patterns. Pattern validation, the process of validating the accuracy of proposed patterns, has rarely gone beyond the simple heuristic of “the rule of three”. This article shows how to use established scientific research methods for the purpose of pattern discovery and validation. The result is an approach to pattern discovery and validation that can provide the same certainty that traditional scientific research methods can provide for the theories they are used to validate. This article describes our approach and explores its usefulness for pattern discovery and evaluation in a series of studies.

---

[1] Corresponding author: Dirk Riehle, dirk.riehle@fau.de, https://oss.cs.fau.de

Table of Contents

1. Introduction

A pattern is the abstraction from similar recurring problem solutions in a defined context (Alexander, 1977) (Gamma et al., 1995) (Buschmann et al., 2007). Pattern authors usually present their patterns in an easily digestible format, the pattern form, and assemble larger sets of patterns into pattern languages, handbooks, or catalogs. As of today, thousands of patterns have been written and published, all claiming to represent common problem solutions.

The process of extracting patterns from expert experience, which we call pattern discovery, as well as the initial evaluation and ultimate validation of proposed patterns or pattern languages has seen little research, however. Pattern authors seek feedback on their patterns in writer's workshops, but such feedback is mostly about the presentation and not about the validity of the presented content. As a consequence, pattern authors have received little to no guidance on how to discover patterns and validate their findings. Pattern authors generally rely on their own experience, and pattern readers have little indication of the validity of the presented patterns but to trust their authors.

In contrast to the patterns community, the scientific research community has developed a large array of methods and approaches to build theories and to validate them. A theory is the knowledge captured about some phenomena of interest; its value or usefulness lies in how well it allows users of the theory to predict the outcome of events and actions. At any point of time, there may be many competing theories to describe the phenomena of interest; these will all be put to the test and over time those theories that fail to predict correctly will be put aside as invalid.

In this article, we describe how to apply scientific methods to pattern discovery, evaluation, and validation. We call the particular sequence of scientific methods that we apply the **handbook method**, a derivative research method. We thereby merge the rigorous work on scientific research methods with the pattern presentation form. Specifically, we show how to use established scientific research methods of theory building to discover, evaluate, and validate patterns. We focus on the domain of software engineering in this article and in the demonstration of our approach, because this is what we are most familiar with.

The contribution of this paper to the patterns community is the following:

1. **The definition and exploration of a start-to-finish process of discovering, codifying, evaluating, and validating patterns.**

This work is in an exploratory stage. The proposed method has been developed by the first author and has been explored in several projects by all authors.

The structure of this article is as follows: After this introduction, Section 2 reviews related work. In Section 3, we describe our approach to pattern discovery and validation using scientific research methods. In Section 4 we discuss several exploratory studies, in which we have applied our approach to demonstrate its usefulness. Section 5 follows with a discussion of this work and Section 6 presents an outlook and concludes the article.

2. Related work

We discuss the two main sources of related work: The pattern community's work on pattern discovery and validation and the scientific community's work on theory building and validation.

2.1 Pattern discovery and validation

Pattern discovery is the discovery of previously not as-such recognized patterns, usually from examples. Pattern discovery is often also called “pattern mining”. Validation of such discovered patterns in the past typically meant applying a simple heuristic, the so-called “rule of three”. According to this rule, a pattern author must present three known (and significantly different) instances of the pattern to substantiate the claim that a new pattern has been found.

2.1.1 Pattern discovery

Iba et al. discuss pattern mining (in the sense of pattern discovery) in a series of articles (Akado et al., 2015), (Iba & Isaku, 2012), (Iba & Isaku, 2016). In these articles, Iba et al. reflect on their experiences with discovering patterns in various domains. Specifically, they use four pattern languages that they wrote as the empirical base for presenting their approach to pattern discovery. Not entirely surprisingly, the approach is presented using the pattern form. Iba et al. perform example processes to illustrate how to discover patterns. They utilize different forms of reasoning during the process.

Iba et al.’s approach to pattern discovery, presented as a pattern language, references their own pattern discovery and development work as examples in which the patterns of pattern discovery have been applied. Their approach consists mainly of creativity techniques, but does not utilize any traditional scientific research methods. Iba et al. apply the rule-of-three heuristic to substantiate their findings. From a scientific perspective, their work therefore remains in proposal status, as the patterns (of pattern discovery) have not been validated using an established scientific method.

In contrast to Iba et al.’s work, our approach utilizes established scientific methods rather than inventing new ones. At this stage, we present a preliminary exploratory evaluation of the validity of the method using three non-trivial studies, leaving a full validation to later work.

2.1.2 Reverse engineering

Outside of Iba et al.’s work, in software engineering, the term “pattern mining”, in the literature typically refers to the identification of existing known patterns in software system design and code. On this topic, Dong et al. (2009) present a survey of known techniques for identifying applied design patterns in source code. Such techniques are used in reverse engineering to unearth previously lost or poorly documented design decisions. Dong et al.’s survey discusses a range of techniques drawn from the literature and we refer the reader to this article to get an overview. In addition to this survey, further design pattern mining articles in the sense of reengineering design decisions have been published, for example, Iacob (2011), Gupta (2011), Gupta et al. (2011), Marco (2012), Alhusain et al. (2013), and Dwivedi et al. (2018).

2.2 Theory building and validation

The work presented in this article puts together established research methods for the purposes of pattern discovery and validation. Pattern discovery is carried out using (mostly qualitative) research methods of theory building, and pattern validation is carried out using research methods of theory validation.

2.2.1 Theory building

Theory building is generally viewed as an iterative process of theory creation and evaluation, where the creation of new theory is followed by (sometimes only partial) evaluation, feeding back into the next iteration of theory creation and evaluation.

Prominent representatives of theory building research methods are:

- **The qualitative survey**, e.g. Jansen (2010). This comparatively simple theory creation method focuses on qualitatively surveying stakeholders and analysing the primary materials gathered using methods of qualitative data analysis, as, for example, defined by Mayring (2000) or Kuckartz (2014). The qualitative survey tries to stay neutral of other research methods and techniques so that it can be combined well, where necessary.

- **Grounded theory,** e.g. Glaser & Strauss (1967), Corbin & Strauss (2014), Charmaz (2006). This heavyweight start-to-finish method of theory building provides a comprehensive set of further methods and techniques of theory building. Arguably, it is better considered a research methodology, forging research methods and techniques into a coherent whole. An important practical contribution by grounded theory is the open coding approach of qualitative data analysis, as discussed by Corbin & Strauss (2014) and as widely used in practice.
- **Case study research,** e.g. Yin (2003), Runeson et al. (2012). This other heavyweight of theory building methods focuses on selecting cases for data gathering and analysis. Unlike grounded theory, which goes broad, case study research can go deep on the usually small number of selected cases. Like grounded theory, case study research is suitable for both theory creation and evaluation, but most authors suggest research should focus either on exploration (i.e. creation) or evaluation of theory, not on both at the same time.

These three methods informed our research. Beyond these three, many other theory building methods exist, for example, action research, e.g. Davidson et al. (2004), ethnographies, e.g. Robinson et al. (2007), and critical theory, e.g. Horckheimer (1972).

2.2.2 Theory validation

In computer science, the predominant epistemological stance is positivistic: Most researchers believe that we cannot only build theories, but can also validate or invalidate them using appropriate research methods. Unlike the iterative process of theory building, theory validation under a positivistic stance consists of a large number of hypothesis tests that probe a formulated theory. Hypotheses are predictions of the theory being tested, and if they turn out to be true, the likelihood of the theory being correct increases, otherwise it decreases. Therefore, each hypothesis test contributes one truth value to the overall validation or (ultimate) invalidation of a theory.

The original research method is the **controlled experiment.** It is often informally equated with “the scientific method”, though this is too simplistic today. The defining characteristic of controlled experiments is that using statistical methods, you can make truth statements about the correlation between input and output variables (independent and dependent constructs), and that this gives definitive answers about the truth value of the hypothesis (within the limits of the experiment’s definition) (Wohlin et al., 2012).

Controlled experiments are a possible approach for testing individual patterns: The pattern’s applied context is the experimental set-up, the problem statement is the independent variable, and the solution is the dependent variable. The challenge, as with most experiments, is to rigorously control for the set-up to avoid confounding factors that influence the outcome.

Another research method of theory validation is the **hypothesis-testing survey** (Fowler, 2013). Unlike the controlled experiment, which goes deep in a specific situation, the hypothesis-testing survey broadly surveys experts or stakeholders as to their thoughts on a particular topic. What may sound subjective is not: Using defined instrument creation and calibration methods, the questions on a hypothesis-testing survey allow for precise statistical analysis and corresponding answers to the underlying hypothesis. As such, the hypothesis-testing survey can be used to broadly query an expert community on the validity of a pattern.

3. Using Scientific Research Methods

This section describes how to use scientific research methods for pattern discovery, codification, evaluation, and validation. We use a running example to illustrate our approach.

3.1 Process overview

Patterns are discovered (“mined”), codified (“written”), and applied, where initial applications often also serve to evaluate and refine the pattern definition. These activities are usually performed in sequence, but eventually also iterated over, until the results seem satisfactory to the pattern author.

Table 3.1 displays the equivalence of these activities. This article introduces the complete cycle, and presents exploratory studies of using scientific methods for pattern discovery and evaluation.

Table 3.1: Terminology and processes of the scientific and the patterns community compared

#	Patterns Community		Scientific Community
1.	**Pattern discovery ("mining")**	↔	**Theory creation**
2.	**Pattern codification ("writing")**	↔	**Theory codification**
3.	**(Reflective) pattern application**	↔	**Theory evaluation**
4.	Proposed pattern	↔	Hypothesis
5.	Pattern testing	↔	Hypothesis testing
6.	Pattern validation	↔	Theory validation

3.1.1 Example

As an example, we will use scientific methods for the development of a patterns handbook of open source governance (Harutyunyan, 2019):

> **Open source governance is the governance of using open source code in a company's software projects and products. Using open source makes the company depend on the code being used, and this dependency needs to be managed. Such governance includes the selection, integration, monitoring and maintenance of open source code and components as well as being compliant with their license.**

Until recently, there were no comprehensive scientific theories nor practical pattern handbooks on how to do this. Some patterns about legal issues of using open source in software products have been published (Hammouda et al., 2010) (Link, 2010), but their discovery has not been documented and they have been presented as self-evident without much clarification of how they were derived.

3.2 Pattern discovery

Pattern discovery is the identification of new patterns, typically from an author's experience. Ideally, the author not only invents the patterns, but rather bases them on occurrences that they have seen.

The first activity for pattern discovery using scientific methods is to use methods of theory creation to identify new patterns and to develop an understanding of their properties in the author's mind. We say theory creation rather than the more general theory building, because theory building in scientific methods not only includes creation, but also evaluation, which we view as a separate activity.

The purpose of theory creation is to create a theory. To do so, the researcher uses one or more research methods. Examples of research methods are the qualitative survey (Jansen, 2010) or grounded theory (Glaser & Strauss, 1967) (Corbin & Strauss, 1998).

Table 3.2 illustrates an example research design for developing a theory of open source governance in software companies.

Table 3.2: Example research process for developing a theory of open source governance

#	Activity	Example
1.	Define research question	What are patterns of open source governance?
2.	Choose research method	Use the qualitative survey (Jansen, 2010)
3.	Write research protocol	Lay-out steps to be taken, assumptions made, ...
4.	Build sampling model	Model companies using dimensions like age, size, ...
5.	Sample relevant population	IBM, MySQL, Bosch, ...
6.	Gather primary data	Interview stakeholders in companies
7.	Analyse data	Perform qualitative content analysis (Mayring, 2000)
8.	Repeat 5.-7. until saturation (stopping criterion) is reached	

With the exception of Iba et al.'s creativity techniques, there are no established methods for the process of pattern discovery that step through the details as illustrated in the research design of Table 3.2. Only scientific methods, here the qualitative survey, come with detailed guidance and textbooks that instruct researchers what to do. This is a significant gain over the pattern community's sole reliance on an author's experience.

Taking this approach, any scientific research method suited to the domain of software engineering can be used, as long as it helps gather the data needed for theory creation. Research methods can be combined, if desired, as long as they don't conflict with each other but strengthen the outcome. The output of such methods must be primary data that can be analyzed.

The analysis of primary data is called qualitative data analysis (QDA) (Corbin & Strauss, 2014) or qualitative content analysis (Mayring, 2000) or qualitative text analysis (Kuckartz, 2014). Common types of primary data are stakeholder interviews, workshop notes, and artifact documentation. The research method defines how to get to this data.

3.2.1 Example continued

In our example of open source governance, we choose

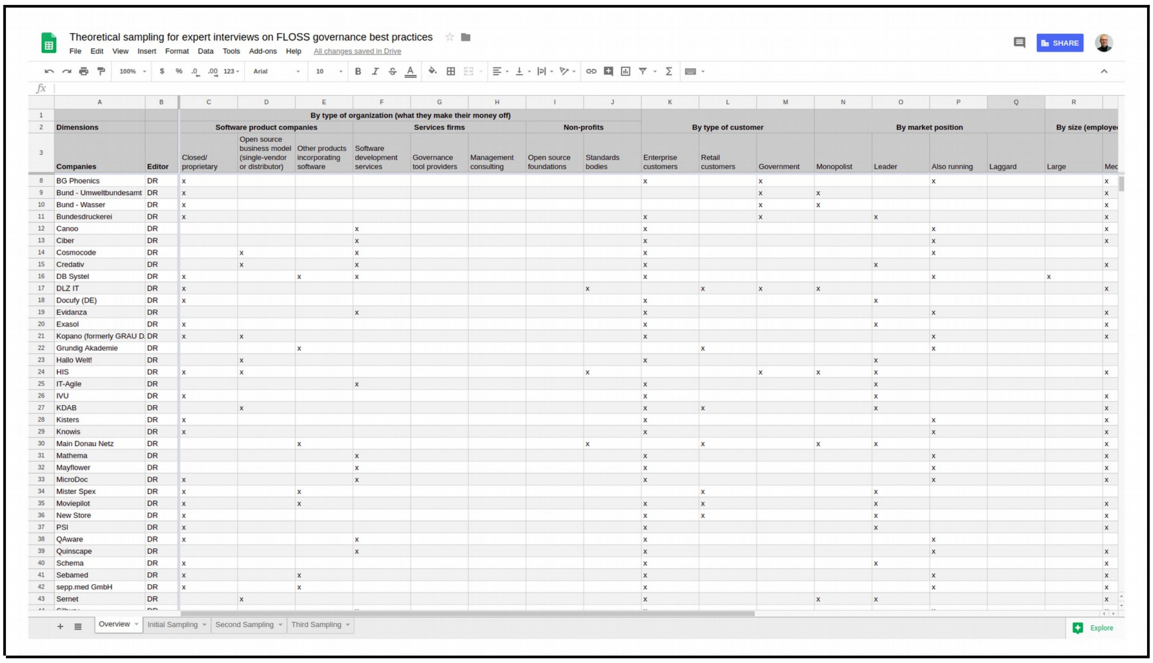
> **What are the patterns of open source governance?**

as the driving pattern discovery question. The underlying assumption is that such patterns exist and can be determined using appropriate theory-building research methods.

Next, we choose the qualitative survey as the simplest applicable research method.

We then determine companies that are experts in open source governance. For this, we develop a sampling model structured by relevant dimensions like size and age of a company, types of products or markets, etc. Using purposive sampling, sufficient coverage of variation can be achieved. Example companies to investigate could be IBM (large, established, traditional products) or MySQL (medium-size, challenger, open source product).

Figure 3.2 presents an excerpt from the sampling, including its model.



Theoretical sampling for expert interviews on FLOSS governance best practices

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1			By type of organization (what they make their money off)								By type of customer			By market position				By size (employe	
2	Dimensions		Software product companies			Services firms			Non-profits										
3	Companies	Editor	Closed/ proprietary	Open source business model (single-vendor or distributor)	Other products incorporating software	Software development services	Governance tool providers	Management consulting	Open source foundations	Standards bodies	Enterprise customers	Retail customers	Government	Monopolist	Leader	Also running	Laggard	Large	Med
8	BG Phoenics	DR	x								x		x			x			x
9	Bund - Umweltbundesamt	DR	x										x	x					x
10	Bund - Wasser	DR	x										x	x					x
11	Bundesdruckerei	DR	x								x		x		x				x
12	Canoo	DR				x					x					x			x
13	Ciber	DR				x					x					x			x
14	Cosmocode	DR		x		x					x					x			
15	Credativ	DR		x		x					x				x				x
16	DB Systel	DR	x		x	x					x					x		x	
17	DLZ IT	DR	x							x		x	x	x					x
18	Docufy (DE)	DR	x								x				x				
19	Evidanza	DR				x					x					x			x
20	Exasol	DR	x								x				x				x
21	Kopano (formerly GRAU D.	DR	x	x							x					x			x
22	Grundig Akademie	DR			x							x				x			
23	Hallo Welt!	DR		x							x				x				
24	HIS	DR	x	x						x			x	x	x				x
25	IT-Agile	DR				x					x				x				
26	IVU	DR	x								x				x				x
27	KDAB	DR		x							x	x			x				x
28	Kisters	DR	x								x					x			x
29	Knowis	DR	x								x					x			x
30	Main Donau Netz	DR			x					x		x		x	x				x
31	Mathema	DR				x					x					x			x
32	Mayflower	DR				x					x					x			x
33	MicroDoc	DR	x			x					x					x			x
34	Mister Spex	DR	x		x							x			x				
35	Moviepilot	DR	x		x						x	x			x				x
36	New Store	DR	x								x	x			x				x
37	PSI	DR	x								x				x				x
38	QAware	DR	x			x					x					x			
39	Quinscape	DR				x					x					x			x
40	Schema	DR	x								x				x				x
41	Sebamed	DR	x		x						x					x			x
42	sepp.med GmbH	DR	x		x						x					x			x
43	Sernet	DR		x							x			x	x				x

Figure 3.2: Example sampling model and population

Within those companies we sample relevant stakeholders, for example, engineering managers, software architects, software developers, the legal counsel, and staff like an open source program officer. We then perform open interviews and ask for supplementary materials like written policies.

With these materials in hand, we start the QDA process and develop the theory. If it becomes apparent that important aspects have not been covered enough, we go back to gathering more materials, either with companies that we already visited or with new companies.

3.3 Pattern codification

Pattern codification is the process of describing a pattern in written form. The presentation of a pattern is often improved through participation in writer's workshops, however, these workshops are usually not supposed to question the content, but only to improve the presentation.

Scientific methods usually have little to say about presentation of theories; they focus on rigorous and traceable derivation of content from primary materials. As such, pattern codification and scientific methods complement each other well.

3.3.1 Data analysis

To go from primary materials like expert interviews to patterns, the material has to be analyzed and the patterns to be derived.

The previous section showed how to gather the primary materials within which the patterns are waiting to be discovered. The next activity in the pattern discovery process is data analysis, from which the patterns emerge. There are many approaches to qualitative data analysis, see the aforementioned ones (Corbin & Strauss, 2014), (Mayring, 2000), (Kuckartz, 2014).

In general, qualitative data analysis consists of annotating the primary materials in such a way that insights emerge incrementally. Initial steps are usually called open coding, because the researcher annotates primary materials with labels ("codes") that represent what seem to be interesting information. Over time, codes are grouped to become higher-level abstractions through axial and selective coding (Corbin & Strauss, 2014) or thematic coding (Mayring, 2000), leading to a multi-rooted tree structure of codes, the so-called code system.

The specific coding paradigm does not matter here, as long as it allows relevant abstractions, here the patterns, to emerge.

3.3.2 Code systems

An important step of going from primary materials to patterns is the creation of a code system.

A code system consists of codes and their sources (codings, i.e. links into primary materials like statements from stakeholders) with associated memos and other materials. It is the result of qualitative data analysis performed during theory building.

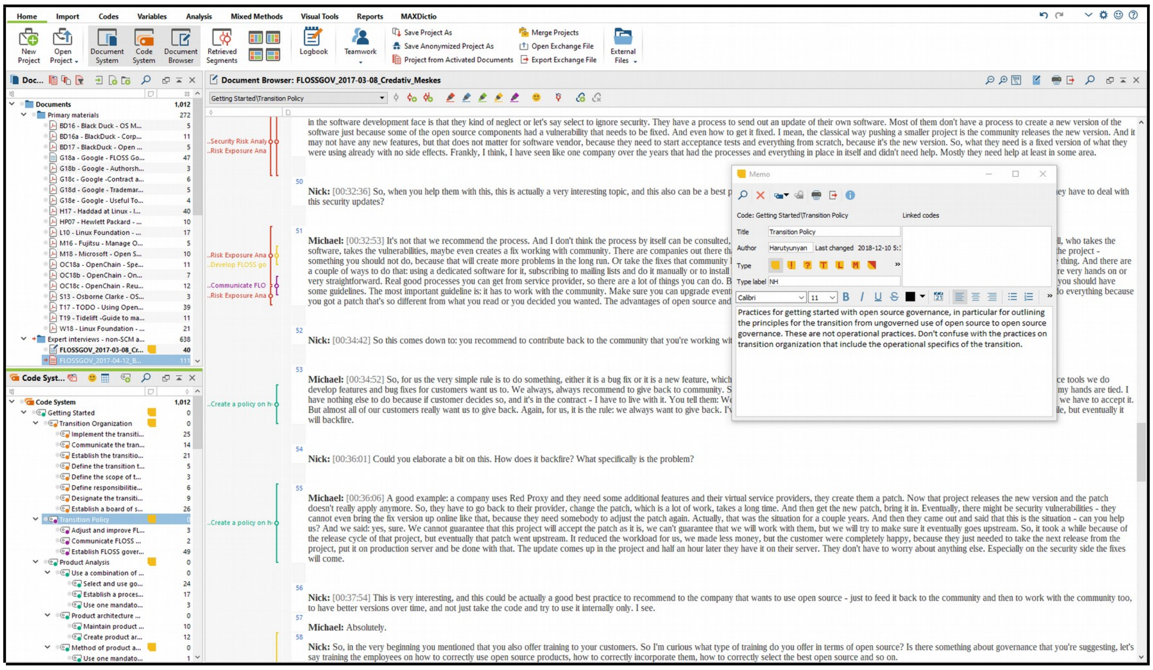
As an artifact, a code system is a hierarchical structure of codes, where each code has zero, one, or more references into the primary materials. Each reference is an annotation of the primary material, and the code is the label of that annotation. Thus, a code can have many instances, called its codings. Codes emerge from the primary material, specifically, when the researcher recognizes something of significance and creates a corresponding code for the first time.

The researcher is usually free in how they structure the hierarchy of codes, but it should follow from their research question and a defined coding process provided by their research method. In general, the higher up in the code system, the more general a code is. Often, a code system has multiple root codes, called core categories, that represent major insights into the theory under development.

Associated with a code is a memo that explains important aspects of the code in general as they can be learned from the codings. Here, the researcher takes notes that can't be represented in the code itself.

3.3.3 Example continued

Qualitative data analysis is often performed using a so-called CAQDAS (computer-assisted qualitative data analysis software) tool. Figure 4.1 shows a screenshot of parts of the code system in our running example together with a memo and some primary material.



in the software development face is that they kind of neglect or let's say select to ignore security. They have a process to send out an update of their own software. Most of them don't have a process to create a new version of the software just because some of the open source components had a vulnerability that needs to be fixed. And even how to get it fixed. I mean, the classical way pushing a smaller project is the community releases the new version. And it may not have any new features, but that does not matter for software vendor, because they need to start acceptance tests and everything from scratch, because it's the new version. So, what they need is a fixed version of what they were using already with no side effects. Frankly, I think, I have seen like one company over the years that had the processes and everything in place in itself and didn't need help. Mostly they need help at least in some area.

50 **Nick:** [00:32:36] So, when you help them with this, this is actually a very interesting topic, and this also can be a best p … ey have to deal with this security updates?

51 **Michael:** [00:32:53] It's not that we recommend the process. And I don't think the process by itself can be consulted, … ll, who takes the
software, takes the vulnerabilities, maybe even creates a fix working with community. There are companies out there th … the project -
something you should not do, because that will create more problems in the long run. Or take the fixes that community … thing. And there are
a couple of ways to do that: using a dedicated software for it, subscribing to mailing lists and do it manually or to install … re very hands on or
very straightforward. Real good processes you can get from service provider, so there are a lot of things you can do. B … you should have
some guidelines. The most important guideline is: it has to work with the community. Make sure you can upgrade event … do everything because
you got a patch that's so different from what you read or you decided you wanted. The advantages of open source and

52 **Nick:** [00:34:42] So this comes down to: you recommend to contribute back to the community that you're working wi

53 **Michael:** [00:34:52] So, for us the very simple rule is to do something, either it is a bug fix or it is a new feature, which … ce tools we do
develop features and bug fixes for customers want us to. We always, always recommend to give back to community. S … my hands are tied. I
have nothing else to do because if customer decides so, and it's in the contract - I have to live with it. You tell them: We … we have to accept it.
But almost all of our customers really want us to give back. Again, for us, it is the rule: we always want to give back. I'v … le, but eventually it
will backfire.

54 **Nick:** [00:36:01] Could you elaborate a bit on this. How does it backfire? What specifically is the problem?

55 **Michael:** [00:36:06] A good example: a company uses Red Proxy and they need some additional features and their virtual service providers, they create them a patch. Now that project releases the new version and the patch doesn't really apply anymore. So, they have to go back to their provider, change the patch, which is a lot of work, takes a long time. And then get the new patch, bring it in. Eventually, there might be security vulnerabilities - they cannot even bring the fix version up online like that, because they need somebody to adjust the patch again. Actually, that was the situation for a couple years. And then they came out and said that this is the situation - can you help us? And we said: yes, sure. We cannot guarantee that this project will accept the patch as it is, we can't guarantee that we will work with them, but we will try to make sure it eventually goes upstream. So, it took a while because of the release cycle of that project, but eventually that patch went upstream. It reduced the workload for us, we made less money, but the customer were completely happy, because they just needed to take the next release from the project, put it on production server and be done with that. The update comes up in the project and half an hour later they have it on their server. They don't have to worry about anything else. Especially on the security side the fixes will come.

56 **Nick:** [00:37:54] This is very interesting, and this could be actually a good best practice to recommend to the company that wants to use open source - just to feed it back to the community and then to work with the community too, to have better versions over time, and not just take the code and try to use it internally only. I see.

57 **Michael:** Absolutely.

58 **Nick:** So, in the very beginning you mentioned that you also offer training to your customers. So I'm curious what type of training do you offer in terms of open source? Is there something about governance that you're suggesting, let's say training the employees on how to correctly use open source products, how to correctly incorporate them, how to correctly select the best open source and so on.

Memo

Code: Getting Started\Transition Policy

Linked codes

Title: Transition Policy

Author: Harutyunyan

Last changed: 2018-12-10 5:

Type label: NH

Practices for getting started with open source governance, in particular for outlining the principles for the transition from ungoverned use of open source to open source governance. These are not operational practices. Don't confuse with the practices on transition organization that include the operational specifics of the transition.

Figure 4.1: Screenshot of example code system, memo, and primary material

3.3.4 Coding process

The data analysis process is usually an iterative process of creating the theory and going back for more primary materials until a stopping criterion is reached.

For this, the researcher works through the primary materials, as discussed. Doing so, they recognize connections between open codes and abstract them into more general codes as part of building a multi-rooted hierarchical code hierarchy, the code system.

The initial observation and the subsequent abstraction through open and axial coding constitute the initial pattern discovery. Since the coding process is iterative, discovering patterns is an on-going process of refining the code system.

Abstract intermediate codes or codes at the root of the hierarchy, commonly called core categories, are prime candidates for patterns. Patterns could be capturing

- whole processes and domains,
- individual workflows and architectural structures, or
- single activities or design structures.

Whatever way a researcher decides to write down as a pattern, eventually, they will always have traceable links to original stakeholder statements by way of codings that justify the pattern at hand.

3.3.5 Pattern handbooks

In our work, we have found a particular structure useful, which we call pattern handbooks. Our handbooks are similar to pattern languages, and we are striving for the holistic generative flow expected of pattern languages, but to avoid confusion with industry we are calling them pattern handbooks.

Table 3.3 presents a mapping of the key concepts underlying a code system and related materials to the different parts of a patterns handbook.

Table 3.3. Code system to patterns handbook mapping

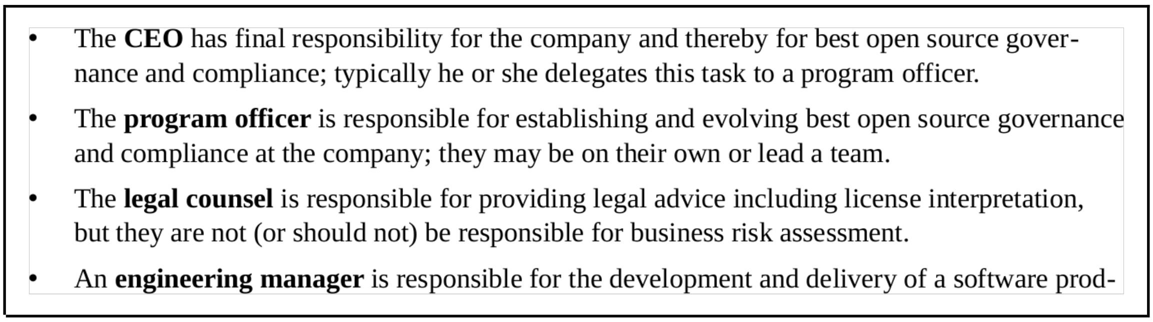
Code System Concept		Patterns Handbook
Core category	↔	Domain chapter
Intermediate or leaf code	↔	Process pattern, practice pattern
Parent/child relationship between codes	↔	Subsections in handbook
Codings	↔	Instances of patterns in primary materials

Summarizing Table 3.3, different codes that emerge as core categories represent different domains in the handbook. Subcodes of the core categories further structure the domains into subdomains and, eventually, process templates and best practice patterns. Codes that represent process templates or best practice patterns should not have subcodes that are different process templates or best practice patterns, but they can have variants of the same process or pattern as subcodes. Codings represent identified instances of the codes in the primary materials.

The current template handbook structure is a reflection of our specific projects and may change over time. Other types of handbooks may have a different overall structure and it is possible that ultimately we will find a way (back) to the classic format of a pattern language as a single graph-like structure of patterns and pattern relationships.

3.3.7 Example continued

Continuing our example of a pattern handbook for open source governance, Figure 3.3 shows an excerpt from our initial role and responsibility section.



- The **CEO** has final responsibility for the company and thereby for best open source governance and compliance; typically he or she delegates this task to a program officer.
- The **program officer** is responsible for establishing and evolving best open source governance and compliance at the company; they may be on their own or lead a team.
- The **legal counsel** is responsible for providing legal advice including license interpretation, but they are not (or should not) be responsible for business risk assessment.
- An **engineering manager** is responsible for the development and delivery of a software prod-

Figure 3.3: Part of a list of roles and responsibilities (short) in open source governance.

Next, Figure 3.4 provides an overview of key domains which structure the handbook as well as some patterns within the domain. We often use a mind-map to visualize the hierarchical structure of domains, subdomains, and patterns, as derived from the code system.

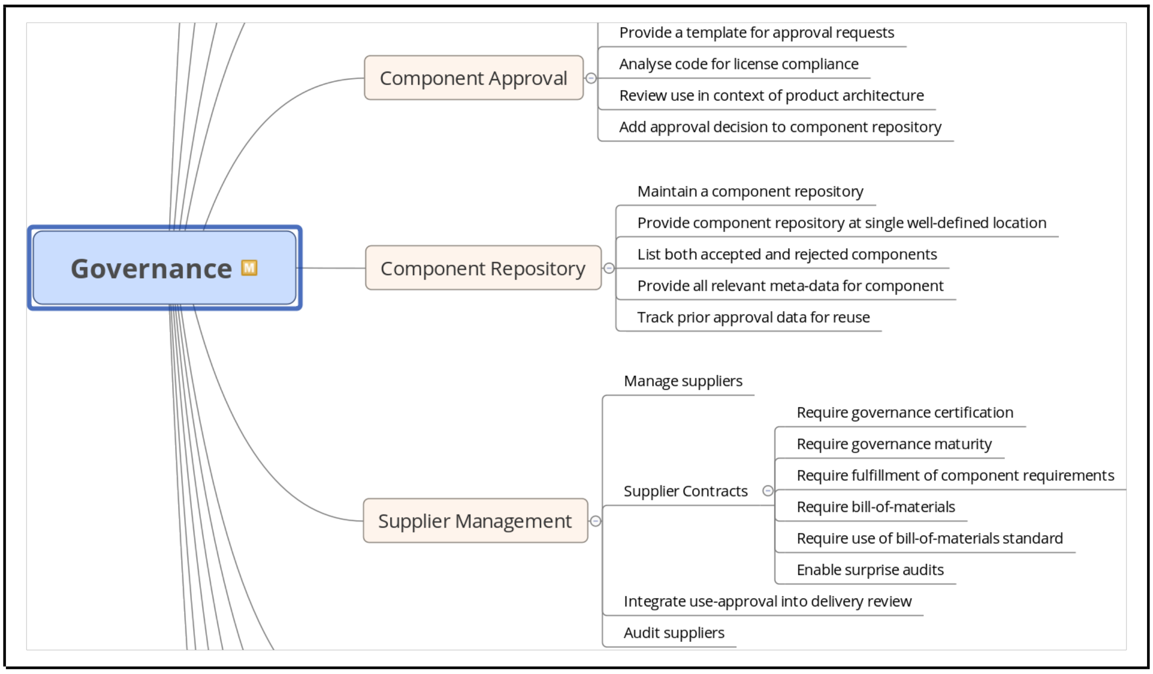


Figure 3.4: Example break-down of domain into subdomains and best practices (patterns)

The hierarchical breakdown of domains and subdomains and patterns as the leaves does not necessarily reflect the way how patterns are linked. Using the handbook by applying patterns one after another happens by following forward references from one pattern to the next.

Within a domain, we have found different types of patterns, for example, those which serve as entry gateways to the domain, those that serve as exit patterns, or pairs of alternative patterns. These types of patterns are in line with prior investigations of writing patterns (Meszaros & Doble, 1997) (Buschmann et al., 2007).

Figure 3.5 shows a short example best practice description in pattern form.

5.7.1 Manage suppliers

Name	Manage suppliers
Actor(s)	Engineering manager
Context	Your product includes not only open source components, but also third-party components that are supplied to you by other software vendors. In contrast to open source projects, you are paying for the component (license) and you are receiving it from a corporate entity. You previously → *defined (your) component requirements* and they must be met by any component, open source or not.
Problem	How to ensure that a third-party component delivery meets your requirements?
Solution	First, before you select a supplier, you may → *require governance certification* or at least → *require (a minimum) governance maturity* of them. Once you have decided for a supplier, in any delivery contract, you should → *require fulfillment of your component requirements* and you should → *require a bill-of-materials* upon delivery for which you → *require they use a bill-of-materials standard.* Upon delivery, you have to → *ensure requirements are met* and for this, you have to → *integrate use-approval into the delivery process.* If the supplier isn't certified and having to reject a component delivery is too expensive, you may want to → *enable surprise audits* and consequently also → *perform surprise audits* as to best goverance practices.
Maturity	Proposed

Figure 3.5: Example best practice, here for managing suppliers in software supply chains

The maturity status describes the scientific evaluation status of the pattern and not the writer's certainty about its presentation quality. The next subsection explains how the maturity status can evolve from proposed to evaluated to validated (or invalidated, that is, rejected).

3.4 Pattern evaluation

Pattern evaluation, as we define it, is the evaluation of the quality of a pattern. There are no methods for pattern evaluation in the patterns community, but one could argue that the actual reflective use of a pattern leads to its reevaluation and rewriting. In fact, some patterns, like the value object pattern (Cunningham, 1995), (Fowler, 2002), (Evans, 2004), (Riehle, 2006), have been written over and over again, with different authors addressing different aspects.

3.4.1 Evaluation vs. validation

In scientific research, there is a difference between theory evaluation and theory validation. Evaluation is usually baked into the theory building process and helps steer the researcher towards increasing the quality of the theory being built. Methods like grounded theory have made theory creation and evaluation complementary but integral parts of the theory building process.

Theory validation, however, is the classic process of making predictions from the theory and testing the resulting hypotheses. Theory building and validation can be kept distinct by the different quality criteria applied to their results. In case of theory building and qualitative work, "trustworthiness" (Guba, 1981) is often used, and in case of theory validation and quantitative work, the traditional reliability and validity metrics are usually used.

Consequently, we distinguish pattern evaluation by application in real-world contexts from pattern validation in laboratory-controlled experiments or similar methods.

3.4.2 Using Case Studies

We have found Yin's approach to case study research useful for the evaluation of our pattern handbooks. Other approaches like action research may work as well, as long as it has been shown that they can be used to evaluate theories, not just to create them.

For this, we apply the handbooks as part of case study research. According to Yin (2013), case study research applies well to phenomena that are

- contemporary,
- occur in real-life, and
- cannot be tightly controlled.

These conditions are a good match for applying our handbooks with their intended audience, for example, software companies. In the following, we walk through the major steps in applying case study research, but keep it short and refer the reader to the original literature.

3.4.2.1 Case study design

For theory creation, the pattern author has to sample experts from which to learn. For theory evaluation, the author samples subjects that want to learn and are open to using patterns for this. Thus, they must not be experts of the domain being investigated. Purposive sampling applies like for expert selection; even the same model can be used. As a rule of thumb, at least three independent and ideally polar cases along relevant dimensions should be selected. In our work, we have only ever used multiple-case designs, embedded or holistic.

The units of analysis depend on the domain, but for software engineering process phenomena they are often the company, specific departments, or the people involved.

The research questions are tied to the effectiveness or "truth" of the handbook and its components. They start with a broad question that gets refined with more detailed questions:

1. Is the domain adequately and completely captured?
2. Is process template X adequate and common?
3. Is best practice Y correct and effective?

To answer these questions, appropriate data need to be gathered. How to do so depends on the pattern handbook's domain and data gathering methods. In general, however, data triangulation is a good idea. Thus, relevant data that can be gathered includes, among others, direct observation, participant observation, and interviews, during or after establishing and applying the processes and practices described by the handbook.

Please note that the research questions are about the actual content of the theory, that is, the practices the patterns describe, and not about the presentation quality of the patterns.

A case study protocol should be written before a specific case investigation is started.

3.4.2.2 Case data collection

Following the case study protocol, the pattern author provides the handbook to the case subjects, e.g. companies, or to whoever needs to apply the handbook as part of the case study.

Usually, the author has to be available to help case subjects in applying the handbook, because there will be many questions. If the author had to help implement the handbook themselves, action research may be a more suitable research method than case study research.

In any case, the author should stay close to the subjects, allowing them to collect necessary data as outlined in the case study protocol.

3.4.2.3 Case data analysis

With data in hand, the author proceeds to evaluate the handbook and answer the questions mentioned above. A challenge will be to separate the following two main dimensions within the materials:

1. Quality of presentation (understandability, applicability, etc.)
2. Quality of content (completeness, correctness, etc.)

The quality of the presentation is the original domain of the patterns community. While intertwined with the correctness of the handbook, it should be dealt with separately from answering the research questions. The data on the quality of content, that is on the correctness and completeness, among other quality criteria, is the main concern, and the research uses it to review the handbook, annotate the maturity of a section, and possibly to trigger a reevaluation and future incremental theory building to improve the handbook.

3.5 Pattern validation

As discussed above, using approaches like grounded theory and case study research for theory evaluation only gives us limited certainty about the validity of the theory (hence our choice of words of evaluation rather than validation). Such broad evaluation is nevertheless useful, because full-blown validation is usually not possible.

To illustrate the point, consider a fully developed patterns handbook with multiple domains, dozens of process templates and hundreds of best practices. Effectively, each element is its own hypothesis. To validate each hypothesis, an appropriate method like a hypothesis-testing survey or controlled experiment has to be applied. This is usually not feasible:

- *For a hypothesis-testing survey,* not only does a survey have to be created, but the respective constructs and their instruments need to be developed first. Given the broad variety of domains and their specificity, we don’t expect that theoretical constructs and their measurement instruments can be used off-the-shelf. Together with a potentially large survey, the amount of work to be performed for validating a full pattern handbook is likely to become prohibitively large.
- *For controlled experiments* to serve as comprehensive validation of the pattern handbook, every single best practice, including its interconnections, needs to be cast as a hypothesis and tested. Given the number of hypotheses and their interactions, this also quickly becomes prohibitively complex and expensive.

In an open world, we can only approach (but never reach) a fully validated theory. Thus, we judiciously choose specific best practices for validation, in such a way that we continuously increase coverage of the theory and incrementally build our trust in its validity.

The set of patterns in a given domain can be viewed as a network of interlinked best practices. Within this network, best practices of a high network centrality are a good choice for hypothesis testing. Examples of such high-centrality best practices are entry and exit (to the domain) best practices. To maximize the impact of hypothesis testing, these best practices should be tested first.

Hypothesis testing, applied this way, buttresses the theory evaluation of other research methods and helps incrementally build trust in the validity of the handbook.

3.6 Incremental process

Sections 3.1 to 3.5 present a rationalized process of pattern discovery, evaluation, and validation. This might suggest a simple linear execution of the work, with resulting patterns at the end. In reality, however, pattern discovery and validation usually proceed incrementally.

We distinguish two different dimensions of incremental pattern discovery and validation:

1. *The incremental discovery of the overall pattern domain structure.* Usually, in a first project, one pattern author approaches the overall domain broadly to develop the initial domain structure. The same author or those who follow them later might incrementally change the domain structure based on new knowledge gathered.
2. *The in-depth development of a particular domain (chapter) in the patterns handbook.* The pattern discovery and evaluation of a particular domain proceeds incrementally, in that after the initial pattern discovery, there usually are learnings gained from the evaluation of the patterns that feed-back to and motivate extended pattern discovery and refinement.

Such incremental development is in line with most theory building methods mentioned so far, in particular, qualitative surveys, grounded theory, and case study research. It depends on the method, however, when another iteration or increment can be started.

This doesn't mean that the work can be approached carelessly and without planning. In particular, if specific aspects like important best practices are selected for in-depth validation, costs can go up significantly. If the tested hypothesis turns out to be invalid, less may have been learned than what was expected and hence resources will have been wasted.

3.7 Industry collaboration

A particularly interesting aspect of our approach is that it lends itself well to collaboration with industry. Let's assume that a particular domain of practice has not been covered well yet by usable patterns as well as scientific theory development and validation. Then, while there certainly might be companies who know how to perform the processes and practices in question well, there will be many other companies who don't know how to do this and have no established body of work to learn from.

This second set of companies, who recognize that they lack some desired capabilities, contains the companies who might be willing to serve as evaluation case studies for the research. They might also be willing to fund the research. These companies would benefit from being given a patterns handbook of process templates and best practices that helps them build the desired capabilities. This is the opening for a principal investigator to motivate a collaboration with these companies.

From our approach's perspective, companies which already possess the desired capabilities can serve as experts for pattern discovery, and companies wanting to acquire the capabilities can serve as case studies for pattern evaluation.

4. Exploratory studies

At this stage of our work, we have formulated our approach and explored it in several studies. These studies are not full-fledged evaluation case studies, but rather exploratory studies to help us understand and refine the approach.

4.1 Overview of studies

Using scientific research methods for pattern discovery and validation has been applied in three exploratory studies and is currently being applied in others more. The three exploratory studies are:

1. *User experience design in product lines.* We applied our approach to eliciting and codifying industry best practices of user experience design for software product lines (Harutyunyan & Riehle, 2019a).
2. *Episodic volunteering.* We applied our approach to eliciting and codifying best practices of managing episodic volunteers in open source projects (Barcomb, 2019).
3. *Open source governance.* We applied our approach to eliciting and codifying best practices for open source governance in software producing companies (Harutyunyan & Riehle, 2019b), (Harutyunyan & Riehle, 2019c), (Harutyunyan, 2019).

The example used in this article draws on exploratory study 3 just listed.

4.2 Evaluation model

To assess the effectiveness of using our approach for pattern discovery, we defined an evaluation model to review the results of the exploratory studies. Table 4.1 presents the evaluation model.

Table 4.1. Evaluation model for effectiveness of using the handbook method for pattern discovery

Quality Criterion	Measurement / Evaluation Metric
Correctness of individual pattern	• Expert found no inconsistencies • Application showed no inconsistencies
Completeness of individual pattern	• Expert found no omissions • Application showed no omissions
Correctness of patterns in domain	• Expert confirmed patterns belong to domain • Application showed patterns belong to domain
Completeness of patterns in domain	• Expert found no missing patterns • Application showed no missing patterns
Correctness of pattern connections	• Expert found no incorrect links • Application showed no incorrect links
Completeness of pattern connections	• Expert found no missing links • Application showed no missing links

Effectively, for three main dimensions (any individual pattern, the patterns within one domain, and the connections between patterns within one domain), we evaluated correctness and completeness. We did so by reviewing the handbook both from the pattern discovery side (asking an expert) as well as the pattern evaluation side (asking users about the handbook application).

Asking an expert meant going back to the experts who we interviewed for pattern discovery. This practice of reviewing the output of pattern discovery is one of the most common quality assessments in theory-building research, called member checking (Creswell & Miller, 2000).

Asking users about the handbook application meant evaluating how well they did in practice by working with our case study partners. At this stage, we are reporting only preliminary findings.

With this evaluation model, we are evaluating the output of three specific pattern discovery studies, and only by extrapolation can suggest that using our approach for pattern discovery actually improves the state of the art. As mentioned, a more rigorous evaluation will have to be done in the future.

Please note that we omitted quality criteria for pattern presentation. While presentation quality is also important, scientific methods have little to say about achieving high presentation quality and we refer the reader to established approaches of the patterns community like writer’s workshops (Coplien & Woolf, 1997), (Gabriel, 2008).

4.3 Study 1: User experience design in product lines

Our first foray into using the handbook method both for scientific research and delivering a practically useful handbook was in the domain of user experience design for software product lines (Harutyunyan & Riehle, 2019a). We conducted multiple-case case study research using two different product lines within the multinational company Siemens AG: In a healthcare software division and in an industrial automation software division. We performed an exploratory study that resulted in a handbook of industry best practices covering the design, implementation, and management of user experience design in the context of software product lines. An example pattern from the study, from the category of UXD Definition, is shown in Table 4.2.

Table 4.2: An example pattern drawn from a study on UXD in product lines

Practice *UXD-DEF-2: Develop SPL-wide templates for new UXD concept definitions, improve them over time and use them consistently.*	
Problem	How to create and formulate new UXD concepts in a detailed, consistent and efficient way across an SPL?
Context	UXD definition is considered a creative process, so definition teams often don't have templates for suggesting new UXD components. Each UXD engineer in the SPL uses the tools he prefers to create UXD concepts and mock-ups, for example PowerPoint presentations. However, often there is a need to compare various UXD concepts in the SPL, which can be difficult, if presentation formats and levels of detail are very different.
Solution	Even though templates are often considered as creativity killers, according to our case studies, if well designed, they can improve the creative process, by stimulating it and putting the necessary limitations and technical constraints in place. The best practice is the development of templates for UXD concepts that would include the technical details of the concepts, its mock-ups and description consistent across the SPL. These templates need to be evaluated and improved continuously to ensure that they are a stimulating tool for the concept development and not another documentation step that is perceived unnecessary and time consuming. The SPL-wide usage of such templates ensures that they evolve and lead to better UXD design. In an SPL such templates will ensure a common approach to the conception of new features and a common UXD definition. Templates save time by avoiding redesigning the basic concept structure every time. This can be a significant benefit. However, the use of the templates for UXD concepts should not eliminate the use of more sophisticated prototypes in the further phases of development. Beyond the UXD concept, there is a need for prototypes, static or dynamic. For these instances, our data suggest a freer approach in terms of the toolset used to formulate the UXD. In these stages the use of templates is not recommended.
Traces in our data:	[Case 1, Interview 3] [Case 2, Interview 1]
Example trace in data:	The head of UXD team from Case 1 explains the practice: "So we have a template for concept definitions. Basically, all UXD changes are done with the use of these concepts. Not only do we define concepts, but there are also some technical aspects to be clarified and there is basically a template that explains what the contents are on the information that needs to be gathered." [Case 1, Interview 3]

4.4 Study 2: Episodic volunteering

We also performed a study on the subject of episodic volunteering in free/libre and open source software communities (Barcomb et al., 2018) (Barcomb et al., 2019). Building on this earlier work, we conducted a Delphi study to determine and confirm best practices. Community managers were asked to describe their concerns about episodic volunteering and the practices they employ to address these concerns. The handbook method was chosen as the best method for relaying the responses to participants to allow for the incremental development of practices over multiple rounds. The final result is a handbook consisting of 65 interrelated practices grouped into five categories: Community Governance, Community Preparation, Onboarding Contributors, Working with Contributors, and Contributor Retention. An example pattern from the study, from the category of Community Preparation, is shown in Table 4.3.

Table 4.3: An example pattern drawn from a study on episodic volunteering

Practice *P.8: Create working groups with a narrow focus*	
Context	The project is too complex for participants to easily comprehend it in its entirety. It is not possible to readily identify stand-alone tasks in the project.
Concerns	● 2.C Episodic contributor lacks awareness of opportunities to contribute
Solution	Create specialized working groups that people can identify with. With a narrow focus and defined outcomes, episodic contributors will be able to find tasks more readily.
Related practices	● *P.6 List current areas of activity* is a possible alternative step. ● *P.18 Write modular software* is a possible alternative step. ● *P.18 Write modular software* is a complementary practice. ● *P.18 Write modular software* is a possible preceding step. ● *O.1 Learn about the experience, preferences, and time constraints of participants* is a possible preceding step.
Challenges	Contributions within the working groups will need to be reported back outside of the group.
Used by	CM2, CM3, CM4, CM5, CM6, CM16
Example trace	“By focusing the working group on a topic that people can identify with, we hope that episodic contributors have an easier time identifying what is useful to them and then have a place to contribute.” — CM4

4.5 Study 3: Open source governance

Our main study, presented here only in its initial exploratory stage (and later as part of full-fledged qualitative survey research), is about open source governance. We define open source governance as a set of processes, best practices, and tools employed by companies to govern the use of open source software components as parts of their products while minimizing their risks and maximizing their benefit from such use. Our work resulted in a theory of industry best practices on the core topics of open source governance in companies:

- Getting started,
- inbound governance,
- outbound governance,
- general governance, and
- supply chain management.

Using the handbook method, we presented our findings in an actionable and industry-friendly format of interconnected best practice patterns that formed a handbook for open source governance. We published parts of the handbook focused on getting started with open source governance (Harutyunyan & Riehle, 2019b) and parts focused on inbound governance (Harutyunyan & Riehle, 2019c). Further parts of the handbook focused on supply chain management were published in Nikolay Harutyunyan’s dissertation (Harutyunyan, 2019). An example pattern from the latter is shown in Table 4.4.

Table 4.4: An example pattern drawn from a study on open source governance

Practice *OSGOV-SUCHMA-BOMMAN-4. Use machine-readable and standard format for BOM upon software supply.*	
Actor	OSPO (Open Source Program Office), Supply chain management responsible role
Context	You have used the bill of materials and code scanning of the supplied code to → *identify open source components and metadata from the supply chain*. You have → *tracked, documented and updated BOM in a consistent and complete manner*.
Problem	How can you improve the performance of managing your BOMs?
Solution	Software supply chains are complex and cannot be handled manually. You need to → *use tools to improve the performance of BOM management*. Most importantly you need to establish a machine readable and standard format for BOMs. An example of such a format is called Software Package Data Exchange (SPDX). It enables the documentation and exchange of data and metadata for open source components and BOMs made of such components.

4.6 Evaluation of studies

As explained in section 4.1, the evaluation model, we used two different methods to assess the quality of our approach with respect to the generated output:

- Member checking
- Case study research

The results confirmed the specific handbooks. Member checking was carried out in all three exploratory studies and domain experts confirmed the individual patterns, the set of patterns, and the connections between the patterns with respect to correctness and completeness.

As mentioned, exploratory study 2 employed the Delphi method, in which a panel of experts over three rounds of questioning and commenting helped us define the practices they knew as patterns. This highly structured elicitation and reviewing process ensures a high degree of confidence in the quality of the patterns that were determined.

Please note again, that such member checking increases our confidence, but does not represent a full-fledged validation due to the inherent biases of asking domain experts flat out to provide feedback.

The exploratory study 3 was evaluated not only through expert member checking but in several full-fledged case studies, using case study research. In this case, the case study researcher observed the use of the handbook within companies which were not experts, and reviewed the patterns, the overall set of patterns, and their connections for correctness and completeness. While we found a few blank spots, the case studies overwhelmingly confirmed the handbook.

5. Discussion

This article presents a new approach for pattern discovery and validation. The key innovation is to use established scientific research methods for pattern discovery and validation in ways that they have not yet been used before.

We also present three exploratory studies that suggest the usefulness of the proposed approach. In addition to the aforementioned three studies, the proposed method was employed to study industry best practices for corporate open sourcing including why and how companies contribute to open source communities (Harutyunyan et al., 2020).

When compared with previous ad-hoc approaches to discovering patterns, our method is both significantly more rigorous and laborious due to its scientific underpinnings. However, such detailed work is justified, and as mentioned, may even be paid for by industry without interfering with any research goal of the studies. It is also highly effective in developing domains where the knowledge exists, but is diffuse. In such circumstances, organic pattern discovery, which depends on an observer encountering multiple examples of the solution, is unlikely to occur.

At this stage of our work, we can only evaluate the trustworthiness of the explorative studies as presented in the previous chapter. The main evaluation method was member checking with the experts whose input was used to create theory and discover the patterns. In addition, for one exploratory study, we performed substantial multi-case case study research.

The evaluation of exploratory research for theory building is not served well by the classic four criteria of test validity (internal validity, external validity, reliability, and objectivity). In response to this unsuitability Lincoln and Guba (1985) defined trustworthiness of qualitative research using four new criteria which they called credibility, transferability, dependability, and confirmability, in analogy to the four criteria of test validity.

- **Credibility.** Credibility is often demonstrated using member checking and triangulation. To make this possible, we would have to give this method to other independent researchers and see them apply it. Then we can check back with them to learn how well the approach worked for them. The second and third author of this article applied the method as conceived by the first author, but because of their relationship (professor and their Ph.D. students), the conclusions that can be drawn from the positive feedback are limited. We argue, however, that the success of the exploratory case studies suggests that the method worked as intended.
- **Transferability.** At this stage, we can’t confirm anything about the transferability of our work to other studies than those presented. However, any break-down in transferability can only come from the particular way of how we put established methods together, because the individual pieces have already been validated separately (as established methods). Our current research will add new studies to the portfolio.
- **Dependability.** When we performed the exploratory studies that provide our preliminary evaluation of using our approach for pattern discovery and validation, our approach itself was only in the process of being formulated. We therefore view potential variation in approach definition as well as potential inconsistencies in their application across the exploratory studies as the biggest possible problem with our evaluation and the quality of the results. However, we did not observe any effects of the evolution of the approach on the evaluation of its effectiveness, as presented in the previous section. We suspect that the reason is that the researchers most of the time simply followed the particular research method at hand, e.g. qualitative survey, grounded theory, or case study research, which are well-defined and have been validated in their own right.
- **Confirmability.** As explained, the approach as presented here is still in exploratory stage. There is no audit trail for the research method but the dissertations of Harutyunyan and Barcomb, and they focus more on their specific research question rather than the approach. As such, we postpone claims of confirmability to the next set of handbooks, currently in work as part of a new set of dissertations. These are being written in a constrained research harness with the purpose of demonstrating confirmability of the approach.

As a final note, we would like to point out that given that there are no other rigorous methods for pattern discovery and validation that utilize established and validated scientific methods, we already improved over the state of the art simply because we provide a new approach for it.

6. Conclusions

This article presents a novel approach, the first method for pattern discovery and validation rooted in scientific research methods itself. We illustrate its usefulness using three exploratory studies.

In future work, we will subject the method to more rigorous testing to confirm that it reliably delivers what it promises, which are patterns handbooks that adequately capture a domain and help the practitioner in solving problems in that domain. Next, we will therefore take a step back and perform theory building and evaluation of the handbook method, for example, by applying the method itself to developing a handbook for using the handbook method.

We aim to provide the patterns community with a rigorous method for pattern discovery and have taken the first step of method definition and exploration with this article.

Acknowledgments

We would like to thank Joseph Yoder for introducing us to Takashi Iba’s work as part of a pattern writing workshop. We also would like to thank Michael Dorner and Julia Krause for participating in a writer’s workshop that helped us improve this article.

References

Akado, Y., Kogure, S., Sasabe, A., Hong, J. H., Saruwatari, K., & Iba, T. (2015). Five Patterns for Designing Pattern Mining Workshops. In Proceedings of the 20th European Conference on Pattern Languages of Programs (p. 9). ACM.

Alexander, C. (1977). A pattern language: towns, buildings, construction. Oxford university press.

Alhusain, S., Coupland, S., John, R., & Kavanagh, M. (2013, September). Towards machine learning based design pattern recognition. In 2013 13th UK Workshop on Computational Intelligence (UKCI) (pp. 244-251). IEEE.

Ampatzoglou, A., Michou, O., & Stamelos, I. (2013). Building and mining a repository of design pattern instances: Practical and research benefits. Entertainment Computing, 4(2), 131-142.

Barcomb, A., Kaufmann, A., Riehle, D., Stol, K. J., & Fitzgerald, B. (2018). Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Transactions on Software Engineering*.

Barcomb, A., Stol, K. J., Riehle, D., & Fitzgerald, B. (2019). Why do episodic volunteers stay in FLOSS communities?. In *41st International Conference on Software Engineering*.

Barcomb, A. (2019). Retaining and Managing Episodic Contributors in Free/Libre/Open Source Software Communities. PhD Dissertation. University of Limerick. Available from https://ulir.ul.ie/handle/10344/8166.

Buschmann, F., Henney, K., & Schmidt, D. (2007). Pattern-oriented Software Architecture: on patterns and pattern language (Vol. 5). John Wiley & Sons.

Charmaz, K. (2006). Constructing grounded theory: A practical guide through qualitative analysis. sage.

Coplien, J. O., & Woolf, B. (1997). A pattern language for writers' workshops. C Plus Plus Report, 9, 51-60.

Corbin, J., & Strauss, A. (2014). Basics of qualitative research: Techniques and procedures for developing grounded theory. Sage publications.

Creswell, J. W., & Miller, D. L. (2000). Determining validity in qualitative inquiry. Theory into practice, 39(3), 124-130.

Cunningham, W. (1995). The CHECKS pattern language of information integrity. In Pattern languages of program design (pp. 145-155). ACM Press/Addison-Wesley Publishing Co.

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. Information systems journal, 14(1), 65-86.

Dong, J., Zhao, Y., & Peng, T. (2009). A review of design pattern mining techniques. International Journal of Software Engineering and Knowledge Engineering, 19(06), 823-855.

Dwivedi, A. K., Tirkey, A., & Rath, S. K. (2018). Software design pattern mining using classification-based techniques. *Frontiers of Computer Science*, *12*(5), 908-922.

Evans, E. (2004). Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional.

Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc..

Fowler Jr, F. J. (2013). Survey research methods. Sage publications.

Gabriel, R. P. (2008). Writers' workshops as scientific Methodology. DreamSongs, Inc. Available at https://www.dreamsongs.com/Files/WritersWorkshops.pdf.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented languages and systems.

Glaser, B. G. (1998). Doing grounded theory: Issues and discussions. Sociology Press.

Glaser, B. G., & Strauss, A. L. (1967). Discovery of grounded theory: Strategies for qualitative research. Routledge.

Guba, E. G. (1981). Criteria for assessing the trustworthiness of naturalistic inquiries. Ectj, 29(2), 75.

Gupta, M., Rao, R. S., Pande, A., & Tripathi, A. K. (2011, January). Design pattern mining using state space representation of graph matching. In International Conference on Computer Science and Information Technology (pp. 318-328). Springer, Berlin, Heidelberg.

Gupta, M. (2011). Design pattern mining using greedy algorithm for multi-labelled graphs. International Journal of Information and Communication Technology, 3(4), 314-323.

Hammouda, I., Mikkonen, T., Oksanen, V., & Jaaksi, A. (2010, October). Open source legality patterns: architectural design decisions motivated by legal concerns. In Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments (pp. 207-214). ACM.

Harutyunyan, N., & Riehle, D. (2019a). User Experience Design in Software Product Lines. In Proceedings of the 52nd Hawaii International Conference on System Sciences. ScholarSpace.

Harutyunyan, N., & Riehle, D. (2019b). Industry Best Practices for FLOSS Governance and Component Reuse. In Proceedings of the 23rd European Conference on Pattern Languages of Programs. ACM.

Harutyunyan, N., & Riehle, D. (2019c). Getting Started with FLOSS Governance and Compliance: A Theory of Industry Best Practices (1:1--1:10). In Proceedings of the 15th International Symposium on Open Collaboration. ACM.

Harutyunyan, N. (2019). Corporate Open Source Governance of Software Supply Chains. PhD Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg. Available from http://nbn-resolving.de/urn:nbn:de:bvb:29-opus4-122727.

Harutyunyan, N., Riehle, D., & Sathya, G. (2020). Industry Best Practices for Corporate Open Sourcing. In Proceedings of the 53rd Hawaii International Conference on System Sciences. ScholarSpace.

Horkheimer, M. (1972). Traditional and critical theory. Critical theory: Selected essays, 1, 188-243.

Kuckartz, U. (2014). Qualitative text analysis: A guide to methods, practice and using software. Sage.

Lau, F. (1999). Toward a framework for action research in information systems studies. Information Technology & People, 12(2), 148-176.

Iacob, C. (2011, June). A design pattern mining method for interaction design. In Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (pp. 217-222). ACM.

Iba, T., & Isaku, T. (2012). Holistic Pattern-Mining Patterns. In Proceedings of the 19th Conference on Pattern Languages of Programs. The Hillside Group.

Iba, T., & Isaku, T. (2016). A Pattern Language for Creating Pattern Languages. In Proceedings of the 23rd Conference on Pattern Languages of Programs (p. 11). The Hillside Group.

Jansen, H. (2010). The logic of qualitative survey research and its position in the field of social research methods. In Forum Qualitative Sozialforschung/Forum: Qualitative Social Research (Vol. 11, No. 2).

Lincoln, Y. S., & Guba, E. G. (1985). Naturalistic inquiry. Beverley Hills.

Link, C. (2010, July). Patterns for the commercial use of open source: legal and licensing aspects. In Proceedings of the 15th European Conference on Pattern Languages of Programs (p. 7). ACM.

Mayring, P. (2000). Forum: Qualitative Social Research. Qualitative Content Analysis, 2-00.

Meszaros, G., & Doble, J. (1997, October). A pattern language for pattern writing. In Proceedings of International Conference on Pattern languages of program design (1997) (Vol. 131, p. 164).

Noble, J., & Biddle, R. (2002, March). Notes on postmodern programming. In Proceedings of the Onward Track at OOPSLA (Vol. 2, pp. 49-71).

OMG (2014). Business process model and notation specification version 2.0.2. Available from https://www.omg.org/spec/BPMN.

Riehle, D. (2006, October). Value object. In Proceedings of the 2006 conference on Pattern languages of programs (Vol. 30). ACM.

Robinson, H., Segal, J., & Sharp, H. (2007). Ethnographically-informed empirical studies of software practice. Information and Software Technology, 49(6), 540-551.

Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). Case study research in software engineering: Guidelines and examples. John Wiley & Sons.

Wagner, S., Mendez, D., Felderer, M., Graziotin, D., & Kalinowski, M. (2019). Challenges in survey research. arXiv preprint arXiv:1908.05899.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer Science & Business Media.

Yin, R. K. (2003). Case study research and applications: Design and methods. Sage publications.

Zanoni, M. (2012). Data mining techniques for design pattern detection. Dissertation, Universita degli Studi di Milano Bicocca.

This page intentionally left blank.