I ♥ Hacker News

Expanding Qualitative Research Findings by Analyzing Social News Websites

Titus Barik
ABB Corporate Research
Raleigh, North Carolina, USA
titus.barik@us.abb.com

Brittany Johnson
NC State University
Raleigh, North Carolina, USA
bijohnso@ncsu.edu

Emerson Murphy-Hill
NC State University
Raleigh, North Carolina, USA
emerson@csc.ncsu.edu

ABSTRACT

Grounded theory is an important research method in empirical software engineering, but it is also time consuming, tedious, and complex. This makes it difficult for researchers to assess if threats, such as missing themes or sample bias, have inadvertently materialized. To better assess such threats, our new idea is that we can automatically extract knowledge from social news websites, such as Hacker News, to easily replicate existing grounded theory research — and then compare the results. We conduct a replication study on static analysis tool adoption using Hacker News. We confirm that even a basic replication and analysis using social news websites can offer additional insights to existing themes in studies, while also identifying new themes. For example, we identified that security was not a theme discovered in the original study on tool adoption. As a long-term vision, we consider techniques from the discipline of knowledge discovery to make this replication process more automatic.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; I.1.m [**Computing Methodologies**]: Miscellaneous

Keywords

grounded theory, computer-mediated discourse, Hacker News, representativeness, theoretical saturation

1. THE PROBLEM

Because human activities play a central role in software development, many research methods borrow from traditional empirical disciplines that study human behavior, such as sociology or anthropology [4]. Motivated by the desire to understand the experiences of actual software practitioners, *grounded theory* is one such empirical research method that has been applied to the study of software development [2].

While researchers disagree on the "correct" execution of grounded theory practices [12], the basic tenet of grounded theory is that insights are generated from collected data, which are then *coded*, or categorized, in a bottom-up fashion to form a theory. In essence, grounded theory is "a method for discovering the real problem that exists for the participants in a substantive area rather than what professional researchers may believe is their problem" [1]. As Brown notes, "the mandate of grounded theory is to develop theories based on people's lived experiences rather than on proving or disproving existing theories" [3]. Put another way, grounded theory provides a systematic and formal approach to ask software practitioners: "What's going on here?"

Unfortunately, grounded theory is also time consuming, tedious, and complex [10] — and like all research methods, has its own requirements, affordances, and limitations. For technical communities who engage largely in quantitative research, three of these requirements cause much angst. First, grounded theory requires that we use *theoretical sampling*, that is, sampling to maximize the diversity of the individuals rather than the size of the sample. Second, the method requires us to reach *theoretical saturation*, or sampling to the point in which no new data appear — that is, a "stop-rule". Third, replication, or other forms of data triangulation, is the recommendation for *transferability* of the theory.

A significant threat is that the grounded theory guidelines for these requirements are presented under some ideal model of unbounded time and infinite constraints — in practice, however, this model is unsatisfiable [8]. For example, sampling may be constrained by physical and geographic factors, which in turn in reduce diversity and trigger theoretical saturation prematurely. And despite its importance, the time consuming nature of grounded theory studies, as well as finite budgets, means that the likelihood that one will actually a conduct a replication is rare [11].

The confluence of these threats makes it difficult for researchers to assess the quality of the *theory* in grounded theory. To address this challenge, this paper investigates the suitability of using posted comments in social news websites as a source of knowledge to mitigate these threats. In the longer term, we propose techniques from knowledge discovery and databases (KDD) and adapt them to grounded theory processes to allow for automatic replication.

2. WHAT IS THE NEW IDEA?

Our new idea is that we can automatically extract knowledge from comments on social news websites, such as Reddit[1],

[1] `https://www.reddit.com/`

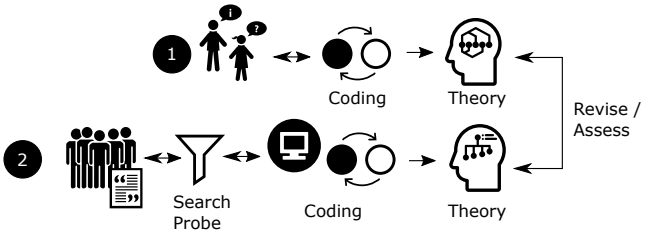

Figure 1: A comparison of the method for (1) a classical empirical studies and (2) a social news website study. In this scenario, a replication study (2) is conducted using Hacker News. Filtering and topic analysis can be automatically or semi-automatically performed. The output of this replication can then be compared with the original empirical study. The double-arrows reflect that grounded theory is not a step-by-step procedure, but an iterative way of thinking about data.

Slashdot[2], and Hacker News[3], to obtain software developer experiences and beliefs for the types of topics that would typically be encountered or asked in grounded theory research. Importantly, unlike surveys or interviews, which require active recruitment to obtain such information, our approach exploits latent knowledge from historical conversations about such topics.

For example, in a technical community, a software engineering researcher might ask the research questions, "Why don't developers use static analysis tools?" [6], "What makes a great software engineer?" [7], or "What are the practices of agile teams?" [5]. We postulate that when such questions are applied against an appropriate social news website, the quality of the results are comparable to the results obtained from empirical research approaches with similar output artifacts — such as transcripts obtained in interviews.

The implementation of our idea can be used to enhance software engineering research in several ways. For example, consider Fig. 1-1, in which an empirical researcher has already conducted a grounded theory study in which he has interviewed participants, coded the responses, and generated a theory. However, the researcher is concerned about the use of student subjects in the study. In our approach (Fig 1-2), the researcher can simply submit a search probe to a social news website using the research question. Because these websites catalog a diverse set of stories, comments relevant to the researcher's probe are likely to exist in the dataset. Using analysis software that implements our technique, the software will extract relevant comments, use machine learning to automatically or semi-automatically code and categorize the responses, and allow the researcher to evaluate the generated theory against the theory derived from the empirical study.

Prior to conducting an in-person empirical study, a researcher may use our technique as a pilot to quickly identify interesting responses to their proposed software-related research question. The researcher can use these results as a tool to guide their coding strategies, by having a set of seed responses. Finally, given a diverse enough dataset, we may also be able to treat this technique as a first-class, stand-alone method for conducting grounded theory research.

[2]`http://www.slashdot.org/`

[3]`https://news.ycombinator.com/`

3. WHY IS IT NEW?

Grounded theory practices have been successfully applied to online software engineering communities to extract useful knowledge. For example, in addition to quantitative analysis, Vendome and colleagues conducted grounded theory on commit notes and issue tracker discussions in GitHub to identify why developers change software licenses [15]. Tsay and colleagues used grounded theory in GitHub to understand how project members evaluate and discuss contributions to the project [14]. Through StackOverflow[4], Nasehi and colleagues used grounded theory on user posts to identify characteristics of effective code examples [9].

However, we are unaware of any research in software engineering that has explicitly attempted to use social news websites to answer multiple grounded theory questions through a generalized technique. Nor are we aware of any efforts to replicate grounded theory studies using online communities.

Our own grounded theory work in understanding why developers use or do not use static analysis tools[5] is the catalyst for the new idea described in this paper [6]. Our interest in this topic is motivated by the experiences in our research lab regarding the cost of conducting grounded theory studies, as well as our desire to increase our own confidence in these types of studies. Thus, our static analysis adoption work is the basis for this paper.

4. A PROOF-OF-CONCEPT REPLICATION OF TOOL ADOPTION

In this section, we conduct a proof-of-concept replication study using our proposed idea. We perform replication to our previous work on why developers don't use static analysis tools [6]. In the original study, we used 20 participants with industry experience. The study session consisted of a semi-structured interview, followed by an interactive interview. A grounded theory analysis was conducted on the transcripts from those interviews.

For purposes of space, we replicate only the primary question from the study, "What reasons do developers have for using or not using static analysis tools to find bugs?" In this replication, we use Hacker News as the underlying data source.

We chose Hacker News for this replication for several reasons. First, it has active discussions, with over 7.5 million comments. Second, Hacker News attracts a technical audience, which means responses to grounded theory questions relating to software engineering are abundant. Third, Hacker News has strict moderation guidelines, as well as comment ranking through a points system, both of which help to enhance the overall quality of comments.

Using Hacker News, we validate our concept through three research questions:

RQ1 Is there enough knowledge available within Hacker News to replicate our study?

RQ2 Are there codes (that is, themes) that we obtain from Hacker News that we failed to identify in our study?

[4]`http://stackoverflow.com/`

[5]Static analysis tools provide analysis on source code, without actually executing the program. They are commonly used for finding defects in code.

RQ3 Can knowledge extraction be conducted automatically?

To answer these research questions, we first filtered all of Hacker News using the search term "static analysis tools". This returned 601 comments by 477 users, from March 20, 2008 to June 4, 2015. For the 78 users who had more than one comment, the mean number of comments was 2.6. Although using this search term fails to retrieve comments that may be related to static analysis but do not specifically use these terms (e.g., `lint`), we felt that the number of returned results were sufficient for a preliminary study. We then randomly sampled a subset of 100 comments (94 users) for our evaluation below.

4.1 Expanding Existing Themes (RQ1)

To evaluate **RQ1**, the first author performed a closed card sort on the comments using the *same* themes as identified in the original study, discarding unrelated comments. The original study contained four themes. One theme, Result Understandability, was derived from the interactive interview. We felt it unlikely that Hacker News would contain any comments pertaining to a very specific experimental task, and as a result, combined this theme with Tool Output. This resulted in three themes: Tool Output, Collaboration, and Customizability.

Tool Output includes comments related to the output produced by the tool. As a result of merging Result Understandability, this theme also includes comments pertaining to the ability or inability to understand or interpret the results produced by a static analysis tool.

As with the original study, we identified the high rate of false positives in tool output as a frustration for developers:[6]

> *[T]he real problem is false positives. If it was caught, but the report was buried in hundreds or thousands of crappy reports about things that actually aren't problems, then it might as well not have been caught.* (7567485)

On the other hand, some tools appear to do a better job than others:

> *I hated every attempt at static analysis until I started programming with Xcode. In my usage, Build and Analyze is always right — that's the difference. Other tools (lint, FXCop) are too noisy.* (4849800)

Our replication also identified an issue that was *not identified* in the original study — the impact of an incorrect fix. For example, two comments in the set relayed the negative experiences when a tool did this. Here's one:

> *I love developing in rails but let's be clear, refactoring ruby is just painful at the moment. Yesterday I decided to rename a model, it took over an hour to get the tests passing again.* (2241462)

Collaboration includes comments about using static analysis tools in a team or collaborative setting.

As with the original study, we identified that tools are used to enforce coding standards within the team:

> *The obvious problem is that if you want to use the C subset in a multi-person project (whose team evolves over time), you have to create a way to enforce that.* (3953726)

Tools are also used in conjunction with collaborative processes, such as code reviews. Tools can guide developers in learning best practices, and, by having tools, the developer can use their time more effectively for higher-level code review issues:

> *The best thing I can say is that when interacting with other people, subtle course correction early on has a big effect in the long run. . . . Code review is an easy one if you're not doing it already . . . as well as doing a little bit of static analysis to automatically detect and fix certain classes of errors. By doing these things, every engineer gets guided to the right path without needing to reach out to anyone.* (5787595)

> *We automatically run FindBugs (along with unit test etc) on our code base every time new code is pushed by a developer and it definitely helps pick up quirky little errors much more cheaply than code reviews.* (1460517)

Customizability includes comments about being able to customize the tool, for example, when modifying rule sets.

Again, as with the original study, we were able to confirm that tool customizability needs to be dead simple, that some developers have no interest in doing even minimal customization, and that the way in which a tool is configured plays a large role in the output that the developer receives:

> *Worse, lint was part of the original C toolchain, but very few people cared to use it, because they couldn't be bored [bothered] to tune the tool to their projects. This is the type of error that any static analysis tool would easily pick.* (7282227)

To our surprise, and unlike the original study, we found that some developers genuinely seem to enjoy customizing their tools extensively:

> *You see, I have file/project trees, warning/error buffers, build status and shells (among other things) in Vim and Emacs with keyboard-driven access to all the tools I use, without having to hunt through a twisty passage of menus . . . [e]ven static analysis and error highlighting is available.* (3548469)

In summary, we assert that **RQ1** is supported, and the use of social news websites can provide additional credibility for themes in existing studies. We also discovered previously unidentified nuances with some of the themes, which supplied additional insights. As a result, we are more confident in the precision of our themes in the original study.

4.2 Discovering Additional Themes (RQ2)

To evaluate **RQ2**, the first author re-examined the comments and performed an open card sort to find comments that did *not* fit into any existing themes. If we are able to identify themes not found in the original study, it would

[6]The number in parentheses for each quote is the object ID in the dataset, for example, `http://hn.algolia.com/api/v1/items/7567485`.

suggest that the original study did not achieve a theoretical saturation that is acceptable for transferability of the theory.

The replication study did in fact identify several themes not discovered in the original study. These themes include awareness (not knowing about a particular tool), cost (regarding the price of the tool), ego (self-evaluation with respect to peers or tools), management (mandates from management to require tool use), maintenance (improving the quality of the code, and supporting legacy languages not designed for static analysis), and security (relating to identifying exploits or other vulnerabilities in the source code). Therefore, we assert that **RQ2** is also supported.

Briefly, let's inspect one of the themes, **Ego**. Even without knowing the individual, we can glean quite a bit of insight about how personality can influence adoption decisions about tools:

> *Unfortunately the decision to use static analysis tools would have to come from developers who are comfortable admitting they make mistakes sometimes. It takes a special kind of ego to write an SSL library with no unit tests, not turn on compiler warnings, and not use static analysis tools.* (7283205)

4.3 Automatic Extraction of Codes (RQ3)

RQ1 and **RQ2** provide support that it is not only feasible, but also useful, to extract relevant knowledge about a software-related research question through the use of social news websites. However, a significant amount of effort, roughly 2.5 hours, was still required to actually perform the coding activities in grounded theory — and thus far we have not discussed any mechanisms to aid the researcher in this effort.

For **RQ3**, we have only begun to examine machine learning approaches to assist with the coding process. One approach to tackling this problem may be to utilize the work of Titov and McDonald, who extend standard modeling techniques such as LDA and PLSA to induce *multi-grain topics* [13]. The unsupervised algorithm is able to cluster topics into coherent, hierarchical concepts. Application of such techniques to grounded theory coding can facilitate automated theory generation. Still other ideas from knowledge discovery and databases (KDD) might be successfully adapted to grounded theory processes, and warrant further investigation.

5. CONCLUSION

Our new idea is an approach in which empirical researchers can leverage social news websites to extract knowledge about software-related research questions that would typically be asked or encountered in grounded theory research. To demonstrate the utility of this approach, we have conducted a proof-of-concept replication of static analysis tool adoption, through Hacker News.

Our approach can be used to augment traditional grounded theory methods in several ways. First, it can extend theoretical sampling, by providing experiences outside of those that may be attainable through physical interviews. Second, it can inexpensively help to assess and identify gaps in theoretical saturation, through relatively simple queries against social news websites. Third, our approach offers a convenient source for replication studies, which aids the transferability of the theory.

With automatic analysis infrastructure, we envision that researchers will not only be able to conduct grounded theory replications to assess the validity of existing grounded theory studies, but also to rapidly understand and theorize about emerging software developer experiences.

6. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under grant numbers 1252995 and 1217700.

7. REFERENCES

[1] S. Adolph, W. Hall, and P. Kruchten. A methodological leg to stand on: Lessons learned using grounded theory to study software development. In *CASCON '08*, pages 1–13, Oct. 2008.

[2] S. Adolph, W. Hall, and P. Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, Jan. 2011.

[3] B. Brown. *Daring Greatly*. Gotham, 2012.

[4] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering. In F. Shull, J. Singer, and D. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, chapter 11, pages 285–311. 2008.

[5] R. Hoda, J. Noble, and S. Marshall. Developing a grounded theory to explain the practices of self-organizing Agile teams. *Empirical Software Engineering*, 17(6):609–639, Apr. 2011.

[6] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *ICSE '13*, pages 672–681, May 2013.

[7] P. Li, A. Ko, and J. Zhu. What makes a great software engineer? In *ICSE '15*, pages 700–710, May 2015.

[8] M. Mason. Sample size and saturation in PhD studies using qualitative interviews. *Forum: Qualitative Social Research*, 11(3), Aug. 2010.

[9] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming Q&A in StackOverflow. In *ICSM '12*, pages 25–34, Sept. 2012.

[10] J. S. Olson and W. A. Kellogg, editors. *Ways of Knowing in HCI*. Springer, 2014.

[11] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo. The role of replications in Empirical Software Engineering. *Empirical Software Engineering*, 13(2):211–218, Jan. 2008.

[12] R. Suddaby. From the editors: What grounded theory is not. *The Academy of Management Journal*, 49(4):633–642, Aug. 2006.

[13] I. Titov and R. McDonald. Modeling online reviews with multi-grain topic models. In *WWW '08*, pages 111–120, Apr. 2008.

[14] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: Evaluating contributions through discussion in GitHub. In *FSE '14*, pages 144–154, Nov. 2014.

[15] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. License usage and changes: A large scale study of Java projects on GitHub. In *ICPC '15*, pages 218–228, 2015.