RepoHunter: Supporting Curation Repositories on GitHub

Y. Wu¹, N. Wang², J. M. Carroll³
¹The Pennsylvania State University
²Samsung Research America

Abstract
In recent years, curation repositories that organize high-quality resources for software development become trending on GitHub. However, few studies have looked at the user experiences of curation repositories, especially what the role GitHub features play in this practice is, and what design efforts can be spent to support this practice better. By surveying software developers about their experiences with curation repositories, this study finds out software developers visit curation repositories to keep themselves in sync with the community (i.e., finding resources for learning, supporting work, and following trends). Information cues on GitHub play important roles for signaling the quality of curated resources. The results inform the design and implementation of RepoHunter, which attaching information cues directly to curated items in a curation repository. The evaluation of RepoHunter reveals the improvement of user experiences and suggests further design opportunities.

Keywords: Curation; GitHub; RepoHunter


Copyright: Copyright is held by the authors.
Contact: yuw132@psu.edu, na.wang1@samsung.com, jcarroll@ist.psu.edu

1 Introduction
GitHub is an online service that provides features for software hosting and collaborating (Dabbish et al. 2012, Marlow et al. 2013). A large community of software developers creates, maintains, and contributes to different repositories on GitHub. In recent years, such repositories are increasingly appropriated as curation repositories, where software developers evaluate, organize, and maintain lists of high-quality resources (Duh et al. 2012, Matthews et al. 2014, Wu et al. 2015). The popularity of curation repositories raises interesting questions concerning why such lists are useful for software developers’ community.

Recent curation literature has investigated curation practices in social networking sites (e.g. Twitter and Pinterest), where researchers find out these curation services serve personal interests (Duh et al. 2012, Zhong et al. 2013). Curation on GitHub is distinct regarding its unique socio-technical environment. First, unlike Twitter and Pinterest, which has a diverse background of general users, curation on GitHub happens within a focused community (i.e. software developers’ community). The unified user body shares common characteristics, such as the motivations to participate in the online community (Von Krogh et al. 20013, Ye and Kishida 20013). Second, GitHub is primarily an online collaborative working environment, which provides unique features for supporting collaboration and communication (Dabbish et al. 2012, Singer et al. 2014), while Twitter and Pinterest are for sharing media contents (e.g., videos, images, and short messages). The unique user body and technical features of GitHub invite research efforts to uncover user intentions and experiences of adopting GitHub for curation purpose.

In addition, adopting GitHub for curation (Wu et al. 2015) is a recent phenomenon, and the practices appropriates a service intended for software development other than curation. It thus invites research effort to explore the limitations and room for improving user experiences regarding curation practices in this particular context.
This paper aims to enrich our understanding of curation and improving curation practices on GitHub. It starts with a survey of 38 software developers, revealing how curation repositories help software developers stay in sync with the community. In addition, popularity, activeness, and project status are the top three types of information cues that are helpful for navigation inside curation repository and evaluation of curated item. Afterward, based on the insights of the survey study, it discusses the design and implementation of RepoHunter, which integrates information cues to curation repository. A user evaluation from 32 participants of RepoHunter finds out its advantages in effective signaling and filtering resources, and limitations regarding user’s preference of quality indicators. This paper concludes with future design suggestions.

2 Background

2.1 Curation Repositories

Curation repositories are regular GitHub repositories, except that they appropriate the README.md file from an introduction page to a list of curated items. Each item includes the resource name, hyperlink, and brief description, as shown in Figure 1. The resources included in curation repositories are manually selected, evaluated, and maintained by software developers. Such lists receive great attention from software developers’ community: many curated repositories become the most starred repositories on GitHub in a short period (Wu et al. 2015).

![Figure 1. The original awesome-go\textsuperscript{1} interface.](https://GitHub.com/avelino/awesome-go, Retrieved on Apr. 13, 2016.)

The broad recognition of curation repositories raises questions concerning user’s experiences with them and their usefulness to software developers’ community. We intend to enrich research literature on curation and software engineering by examining and improving curation practices on GitHub.

2.2 GitHub Transparency
In social media related research, transparency refers to the visibility of others’ actions on public or shared artifacts (Dabbish et al. 2012). It also includes social signals, such as the number of votes associated with an answer or the number of followers a user has (Storey et al. 2014). On GitHub, transparency, such as the number of stars and forks of a GitHub repository (McDonald and Goggins 2013), and the visibility of developer activities and profiles, are crucial, because it can motivate others to contribute, and thus shape the success of a repository (Dabbish et al. 2012, Wu et al. 2014). For example, software developers evaluate the frequency of contributions and the involvement of a “coding rock star” of a repository to decide whether a repository is worth contributing to (Dabbish et al. 2012).

In a recent study, researchers find that software developers actively seek information about each other, and use these pieces of information to form impressions about peers and project status, which in turn guide future interactions and affect the trustworthiness of a contributor (Marlow et al. 2013).

These works answered important questions on the usefulness of transparency for software developers to make connections and contribute to others. However, the role of transparency plays in curation practices on GitHub is not reported.

2.3 Information Cues
GitHub transparency is embodied by information cues. In an information environment, an information cue is a signal that can influence users’ evaluation of information sources (Pirolli and Fu 2003). For example, in news browsing context, news source, time elapsed since the story, and the number of articles that refer to the news are information cues that can influence users’ perception of a news item (Sundar et al. 2007). The transparency of GitHub provides abundant information cues. For instances, the number of followers signals whether a user is a “coding rock stars”, and the number of code commits suggests the activeness of a repository (Dabbish et al. 2012). Existing research suggests these cues lead users form attitudes and make evaluation about information on the site (Kim and Sundar 2011).

Given the prevalence of information cues on GitHub, it raises interesting questions about what information cues software developers look for when browsing curation repositories, and how they are useful.

3 Study1 – Understanding Awesome-go Curation Repository
To gain a better understanding of curation repositories on GitHub, we conducted a qualitative study to collect users’ feedback to the awesome-go curation repository (Figure 1), one of the most popular curation repositories when we sent out our survey in April 2016. The awesome-go repository was created in July 2014, and it now has more than 16,000 stars, which is within the top 50 most starred repositories created after 2014. It indexes resources about the Go Programming language. As we considered collecting detail and unexpected feedback from responses, we utilized open-ended questions to give respondents the freedom and space to give any answer, of any length and with any level of detail. In this way, respondents will not be confined to a limited set of answers that are available in a Likert-type design.

3.1 Participants and Procedure
We designed a web-based open-ended survey study using the Google Form survey platform. Our recruitment began in April 2016 and completed in the same month. To first generate a list of potential participants, we called the GitHub API and fetched profile information of software developers who had starred awesome-go project. We then randomly sent out survey invitations to 800 awesome-go followers. There were in total 38 participants who consented to partake in our study.

---

Participants were asked to respond to open-ended questions about their feedback towards the awesome-go project from various perspectives. The survey questions were guided by the following themes:

- What are software developers’ motivations to visit the awesome-go project?
- What types of resources do software developers find most useful?
- How GitHub transparency supports evaluating resources in a curation repository?

Respondents are allowed to skip questions.

The average professional working experience for our participants is 6.2 years. 9 participants have high school degree; 21 have a Bachelor’s degree; 7 have a Master’s degree; and 1 has a doctorate.

After data collection was completed, we analyzed the responses to identify key findings. We performed the qualitative analysis of answers with open coding schema (Strauss 1987). Codes were discovered, discussed, and refined among the authors.

3.2 Data Analysis and Results

The response data was coded based on a set of rules developed from the questions asked, as well as information received from the responses. As the first step, we examined participants’ responses to the open-ended questions and identified significant concepts and aspects through content analysis of their answers. We then applied a more thorough process to code their answers for in-depth analysis. In this procedure, we examined the patterns of participants’ responses and attempted to correlate concepts generated in the coding process. By doing this, we expected to better understand participants’ experiences with curation repositories on GitHub. More importantly, we expected these results could infer a better curation repository design in practice.

Our analysis of software developers’ experiences with curation repositories on GitHub are unfolded in the following steps.

3.2.1 Reasons for visiting curation repositories

The main reason that software developers visit curation repositories is to sync with the community, including following the trend, learning, and problem-solving.

Seven mentioned that they wanted to keep themselves updated about the development topic, such as “I wanted to monitor what go projects are respected by the OSS community”. 16 participants reported that they intended to find learning materials, i.e. tutorials and best practices about Go Language, e.g. “I’m fond of Go, and one of the ways to improve myself was to learn from the existing go-projects, if not the best.” And ten participants tried to find resources and solutions to achieve specific development tasks, for instance, “Sometimes I need to look for a specific tool or anyway to get an idea of available projects in a specific domain without spending the whole day with a search engine.” In addition, 18 participants expressed general interests towards the topic, such as “(I have) huge interest in go lang resources, and it is a nice list of curated resources.”

These reasons are comparable to software developers’ reasons to visit other social media services. For instances, software developers visit StackOverflow to find learning resources and code examples (Nasehi et al. 2012)). They visit Twitter to stay up-to-date with latest technologies, tools, and changes (Singer et al. 2014). And they find open source projects to use and learn from on GitHub (Dabbish et al. 2012, Wu et al. 2014).

To be more specific, curation repositories take a different approach for software developers to keep up with the community: they centralize otherwise distributed resources. Besides, the resources included in curation repositories have been evaluated by others peers, filtering out low-quality resources. So instead of jumping from one site to another and filtering and evaluating resources themselves, software developers can visit curation repositories for peer-reviewed high-quality resources.
3.2.2 Information cues for evaluating curated items

When asked what types of resources they found most useful during their visits of the awesome-go repository (33 responses), the top two categories were software repositories (27 mentions), and text about the development, i.e. articles, books, and presentations (10 mentions). Notably, three participants mentioned that they were mostly interested in software repositories that were hosted on GitHub. Given text resources are usually located outside GitHub where the information environment is not standardized, in this study, we only investigate information cues for software repositories on GitHub.

Specifically, we asked our participants to answer what signals, cues, and factors they would consider important and helpful when evaluating curated resources in awesome-go. 32 participants responded to this question. The answers can be grouped into three categories (Table 1) – popularity, activeness, and project status. For instances, the number of stars signals popularity, the last pushed/commit date implies the activeness of a project, and the issue list indicates whether the project is used by others or has bugs currently.

<table>
<thead>
<tr>
<th>Information cues category</th>
<th>Available cues</th>
<th># of mentions</th>
<th>Example quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Popularity</td>
<td>The number of stars/forks</td>
<td>25</td>
<td>“starts are quite good indicator that people at least are interested with the project (must be a reason for that).”</td>
</tr>
<tr>
<td></td>
<td>Last pushed date, the number pull requests</td>
<td>14</td>
<td>“First of all I exclude dead projects whose last commit date was around at least 3-6 month before.”</td>
</tr>
<tr>
<td>Project status</td>
<td>Documentation, the issue list</td>
<td>5</td>
<td>“I tend to exclude projects with poor documentation.”</td>
</tr>
</tbody>
</table>

Table 1. Information cues that software developers look for when evaluating curated items.

Although these information cues, i.e. the number of stars and last updated date, are available in each repository, as shown in Figure 1, they are not contained in curation repositories. Thus, when a software developer wants to check out the statuses of many software repositories, they have to open each in a new tab, and switch back and forth. It becomes notably inconvenient when software developers try to compare several repositories.

In summary, curation repositories present an innovative way for software developers to keep themselves in sync with the community by centralizing high-quality resources distributed in different places. Although the resources in a curation repository is manually curated, software developers still look for additional information cues when evaluating each curated item inside the repository. Due to the abundant resources inside each curation repository, as well as the lacking of direct information cues, curation repositories create usability issues.

In the following sections, we will discuss our design and implementation of a prototype to address these issues, and present a user evaluation of the prototype.

4 Building RepoHunter to Bring Information Cues to Curated Items

The results suggest that the major drawback of the current design of curation repositories is the extra effort to find information cues for each curated item. To address this issue, we designed RepoHunter, which brings information cues to each curated item to make them readily available.
Continuous Integration

Tools for help with continuous integration
- **drone** ★ 6715, pushed 1 days ago - Drone is a Continuous Integration platform built on Docker, written in Go
- **govertis** ★ 250, pushed 37 days ago - Go integration for Coveralls.io continuous code coverage tracking system.
- **overallis** ★ 18, pushed 215 days ago - Multi-Package go project coverage profile for tools like govertoris

CSS Preprocessors

Libraries for preprocessing CSS files
- **c6** ★ 339, pushed 196 days ago - High performance SASS compatible implementation compiler written in Go
- **gcas** ★ 340, pushed 582 days ago - Pure Go CSS Preprocessor.
- **go-libsass** ★ 26, pushed 28 days ago - Go wrapper to the 100% Sass compatible libsass project.

Data Structures

Generic data structures and algorithms in Go.
- **binpacker** ★ 12, pushed 71 days ago - Binary packer and unpacker helps user build custom binary stream.
- **bitset** ★ 200, pushed 61 days ago - Go package implementing bitsets.
- **bloom** ★ 82, pushed 183 days ago - Bloom filters implemented in Go.

Figure 2. The *RepoHunter* version of the awesome-go, including the number of stars and last pushed date for each curated item.

4.1 Design Details

The purpose of *RepoHunter* is to make it efficient and convenient for users to visit curation repositories. We intended to explore if drawing cues directly to curated items could be helpful for software developers. Specifically, we brought information cues of popularity and activeness to each GitHub repository in a curation repository. Figure 2 illustrates the interface of *RepoHunter* for the awesome-go repository.

Stars Count - Popularity Cue

“Star” is a GitHub feature that allows software developers to bookmark a repository for future reference (Dabbish et al. 2012). The number of stars demonstrates how many software developers show interests to a repository, and thus it is a good indicator for popularity (McDonald and Goggin 2013). We put the number of stars of a curated software repository right beside the resource name in *RepoHunter*.

Last Pushed Date – Activeness Cue

Our previous survey study shows that last pushed date an important information cue for activeness, and the reason is that software developers want to avoid projects that are no longer maintained. We put the last pushed date of a repository beside the number of stars (Figure 2).

The reason that we excluded project status, another type of information cue as discussed above, is because the information is not readily available either from each repository or through GitHub API.

4.2 *RepoHunter* Implementation

*RepoHunter* is implemented in the following steps: 1) it retrieves the contents of README.md of a curation repository, converts it into an HTML file; 2) it scans the HTML file, and for each curated item that is a GitHub repository, it queries the GitHub API to retrieve the current number of stars and last pushed date; 3) *RepoHunter* then attaches the info to the corresponding curated item, and then 4) regenerates the README.md file with the modified HTML contents. The generated file is also put on GitHub so that it contains nearly the same interface and environment as compared to the original curation repository, except for the additional information cues.
4.3 Preliminary User Evaluation
Following the similar procedure, we randomly sent survey invitations to 800 GitHub users who starred awesome-go repository, requesting them to use RepoHunter, and to compare it with the original awesome-go interface. We concerned about whether RepoHunter would help them achieve their goals for visiting awesome-go. 32 responded to our survey. The same data analysis procedures in our first user study were applied. (The results section are simplified due to words limitation.)

4.4 Results
General Attitudes toward RepoHunter
The advantages of RepoHunter regarding efficiency and convenience were confirmed by participants. 27 out of 32 participants highlighted the improvements of RepoHunter. For example, participants commented the following:

“[RepoHunter] is better. It contains more details”

“[RepoHunter] gives me more information faster and let’s me judge the project’s long term strength. [RepoHunter], by a landslide.”

“I prefer [RepoHunter]. That shows clearly the repo status we concern about”

5 out of 32 participants preferred the original page. 3 of them thought the extra information was not good indicators of quality, and 2 of them considered the number of stars and last pushed date noise, which didn’t help them evaluate curated items.

Perceived Effectiveness of Information Cues
Participants regarded the new RepoHunter design as superior in terms of effectively providing aggregated information in one page. Both star counts and the last pushed date were highlighted:

“It would save me some time since I will not have to open several tabs to check the stars of each package, which is an important thing when I’m looking for a package.”

“Low stars/low activities are not interesting to me. Helps me filter quickly.”

“...[I] really like the most recently pushed.”

Future Interface Improvement
Participants also pointed out some directions to improve the format and layout of RepoHunter in the future. For example, participants suggested structuring the information into a table, and put the extra information in a separate column: “It needs table structure now when there’s too much info for one line.”

5 Discussion and Conclusion
The evaluation of RepoHunter yields promising results. Software developers find integrating information cues provides better experiences for visiting curation repository; they are helpful for filtering out unwanted items, and for comparing similar options. In the meantime, our analysis also suggests further investigation.

Software developers take information cues as indicators for software quality. While most software developers are satisfying with popularity and activeness cues as a proxy for quality, some others are not. Our future endeavor will include in-depth understanding the evaluating and decision-making process for evaluating curated items and iterate different quality cues, to best suit software developers’ needs for visiting curation repository.

In summary, this research seeks to add to the research literature by providing a greater understanding of curation repositories on GitHub. The development of design principles as well as the preliminary evaluation of our proposed RepoHunter could improve curation practices on GitHub. In the
future, we intend to apply our proposed RepoHunter framework to other popular curation repositories on GitHub and other platforms to provide the iConference research community with comparable results across a spectrum of choices. We also would like to explore opportunities for future collaboration with GitHub and software developers to conduct large-scale field experiments in the context of naturally occurred user practices.

6 References


