Christopher Addison (cjteam)
William Cao (pastor13)
David Dupre (djdupre)
Jared Le (jaredtle)
Christine Ta (cta95)

Track Forever

Motivation

Issue tracking software is essential to any team's development process. For instance, they help teams record and organize software bugs, and assign tasks to team members. There are many tools available for issue tracking today (see Competitors section). However, there are limitations that can hinder a team's ability to efficiently track their bugs. For example, when services are discontinued (e.g. Google Code), a team's existing issue entries may become difficult to access.

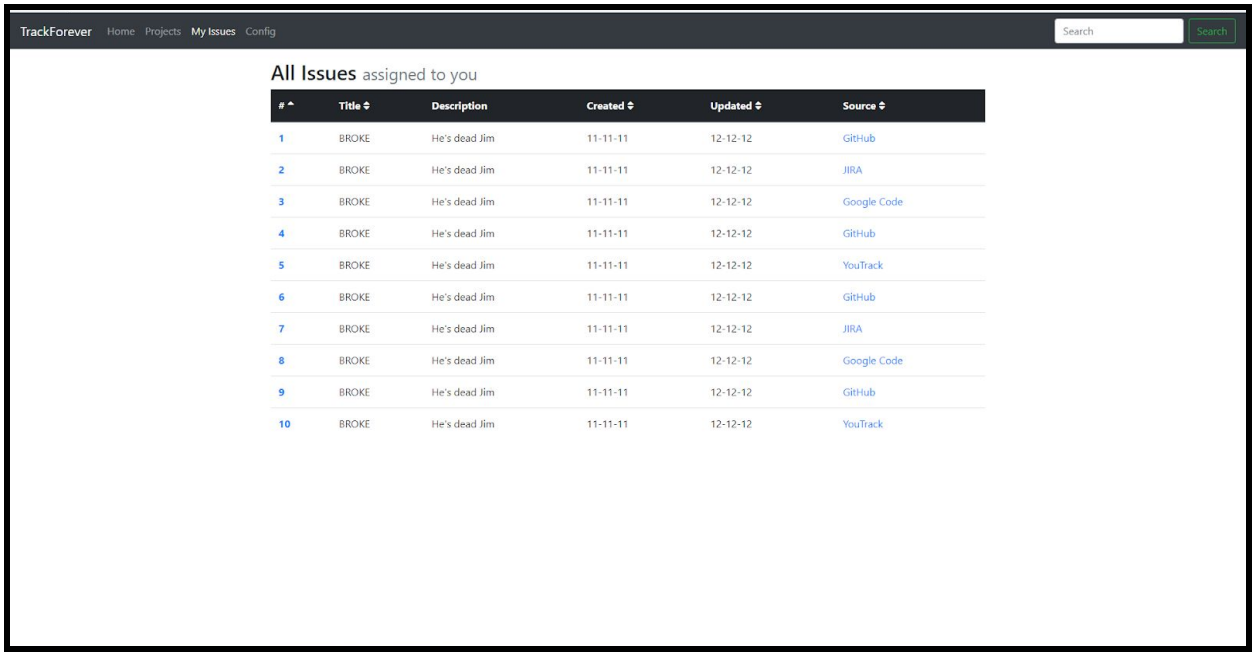
The primary goal of Track Forever is to future-proof a great number of issue trackers so that when an issue-tracking service is discontinued, the existing issues remain accessible. Creating a tool to migrate issues from many issue trackers to one, unified file format will be instrumental to this goal. In addition, Track Forever will also offer offline access to tracked issues across any number of projects and services through the use of a portable user interface. Finally, Track Forever may assist users in handling migrations between different issue trackers.

Approach

We will first create an open-source unified issue tracker storage format that combines the common features across other issue trackers. We will also create a conversion tool that will use existing issue tracker APIs to download tracked projects and convert the results into our unified file format. This format will include fields shared by the most popular issue trackers we plan to support. Those fields are, at the very least, summary, status, project, assignees, submitter, and tags.

Although we initially plan to build our implementation to match the Google Code Archive API, we will extend our program to include compatibility with popular issue trackers that have similar API support. For example, we could start with some of the several popular issue trackers like Jira and YouTrack that offer a RESTful API. This approach will allow code reuse between the support of various issue trackers. We will also focus on building modularity so that other contributors can easily implement their own conversion plugins, allowing for quick conversion to our unified file format. This can potentially assist those migrating from issue trackers that risk being discontinued.

With the unified file format we will create a webapp similar to GitHub's issue tracker to view issues offline (see fig. 1 below). The website will at first only support viewing, but could over time become its own usable issue tracker.



(fig. 1) Issue tracking website mockup

Competitors

There are plenty of issue trackers in use today. In modern iterations of this type of software, the backbone is typically supported by a popular database program. The front-end implementation of these issue trackers vary greatly, although many use webapps. Most of these issue trackers, however, are not accessible offline. This can be an inconvenience for developers that currently do not have a network connection as they will be unable to look up current issues to work on them. Our approach is to fill in the usage gaps left by these other programs, allowing for offline issue viewing and better conversion between other issue trackers.

Some other approaches to this problem have good integration with other issue trackers but are not powerful enough. For example, Taskwarrior is a promising open-source project but it does not support collaboration, run on Windows, or present a user-friendly interface. We hope to make this project into something that supports teams of developers, greater accessibility regardless of operating system, and an intuitive interface that users will enjoy. There are also proprietary, enterprise approaches to unified issue trackers, but they do not have as broad integration with other issue trackers. Even though some issue trackers can export issues into various file formats (.csv, .xml, and .json), there is no universal standard that these issue trackers agree on making it difficult to migrate from one issue tracker to another. By creating a file format that is open-source, we hope that integration will come more easily as more contributors join the project.

Eventually, as our project progresses, we can begin to optimize and address problems that affect issue trackers. For instance, research papers from within the last five years suggest that a number of issues submitted into modern issue trackers were actually duplicates[1]. We could improve the webapp with a reminder that appears before a user submits an issue to check or search for possible duplicates or related issues that have been previously submitted, as this is shown to decrease the number of duplicate issues within a project[2].

Challenges and Risks

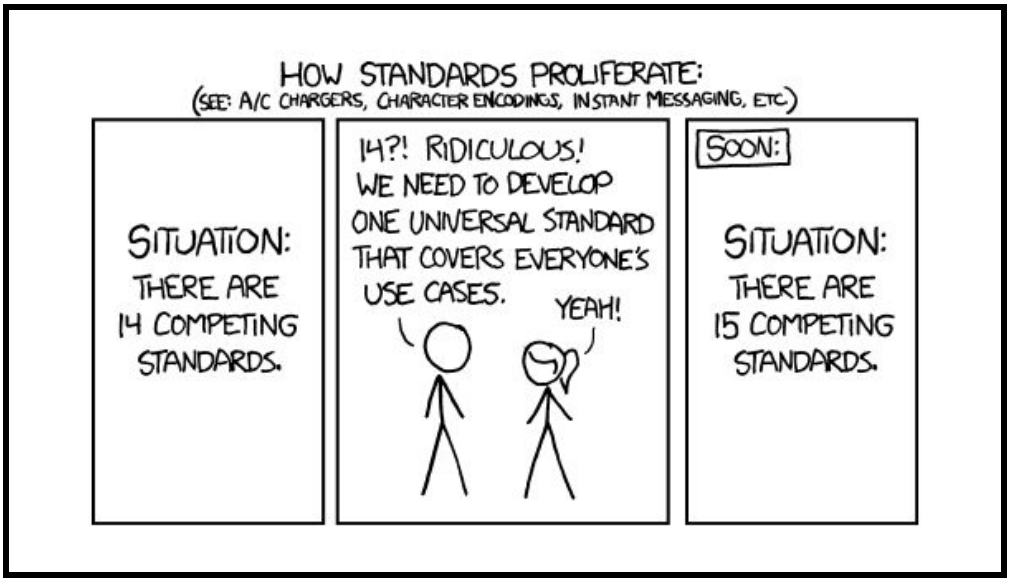
There is always the risk that an attempt at a “unified” standard becomes just another competing standard (see fig. 2 below). For this tool to stand out above others, we need to support as many issue trackers as possible. We hope to overcome this risk by prioritizing tools for adding support for other issue trackers. This will enable ourselves and other contributors to easily integrate other issue trackers, increasing the number of use cases (and therefore our project’s appeal). By building a simple format for tracked issues that is strictly independent of our program, we will also allow others

---

[1] Gu, H., L. Zhao, and C. Shu. *Analysis of Duplicate Issue Reports for Issue Tracking System*, 2011, *SCOPUS*, www.scopus.com.

[2] Petra Heck and Andy Zaidman. 2013. An analysis of requirements evolution in open source projects: recommendations for issue trackers. In *Proceedings of the 2013 International Workshop on Principles of Software Evolution* (IWPSE 2013). ACM, New York, NY, USA, 43-52. DOI=http://dx.doi.org/10.1145/2501543.2501550

to build their own implementations of the issue viewer, encouraging participation and growth in the project.



(fig. 2) A well-known comic, xkcd #927

Another challenge will be adapting issue trackers to fit into one format. Although most issue trackers share fields such as title, number, and description, many have custom fields and text features that are difficult to reimplement, let alone maintain across a conversion. In some cases, mapping between each custom field and our own implementation can be made for each conversion module. However, in some cases we cannot make the translation, and some information may be lost.

Metrics

There are several ways we can measure our progress. One such way is to pick an issue tracker like Github, finding the top 10 open source projects, and importing issues from these projects. Afterwards, we can calculate the percentage of issues that were successfully imported from these projects. Based off this percentage calculated, we can generalize this to all of Github and make claims about the approximate number of issues compatible with our issue tracker within Github. Another method of measuring progress is by taking the sum of the market shares of the issue trackers supported by our project. For instance, once we can get our issue tracker working with a popular issue tracker like Jira, we can say that we support around 48% of the market share of issue trackers. When talking about the speed of our issue tracker, we can talk about the average number of issues that are converted by our program per second.

Schedule* and Milestones

Week 1 - Project proposal

✓ Project pitch and team formation

Week 2 - Project proposal

✓ Set up team tools and github repository
✓ Refine project proposal by reviewing initial ideas and approaches, conduct further research on market and competitors, challenges, and goals
✓ Create a weekly project schedule and a scrum-like board for task management

Week 3 - Design and planning

- Get and understand all file formats of bug entries from *at least* two issue trackers (considerations include Google Code, Jira, and YouTrack)
- Design a unified file format for our issue tracker
- Design the backend system architecture
- Design API contracts for interfacing between web app and backend
- Design the UI for the web app

Week 4 - Implementation begins

- Create a conversion tool for two bug trackers to our own format
- Implement the UI of the web app using dummy bug entries
- Research tools to allow offline support for our web app (i.e. a service worker)

Week 5 - Implementation continues

- Create a conversion tool for at least one more bug tracker
- Implement the REST API for interfacing between front and back end
- Implement offline support in web app such that the page remains even after refresh
- Revise project proposal

Week 6 - Implementation continues

- Configure web app to call API and display bug entries
- Re-revise project proposal

Week 7 - Implementation continues, testing and QA

- Write initial project results
- Ensure that automated test suite is operational and integrated into build system

Week 8 - Testing and QA

- Continue testing, polishing, and bug fixing. Also consider optimization.
- Prepare for initial project presentation

Week 9 - Testing and report writing

- Finalize test suite, ensuring as many bugs as possible are exterminated
- Begin drafting of final report, also begin measuring performance statistics
- Complete instructions in repository that allow for reproduction of experimental results

Week 10 - Testing and wrap-up

- Automate instructions for reproducing experimental results
- Code review one last time
- Create final presentation slides

Week 11 - Final presentation and report

* Schedule is subject to changes, revisions, modifications, alterations, and adjustments