

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. Though there are many tools available for issue tracking today (see Competitors section), there are common 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. When this happens, the developers must now spend their valuable time porting issues from a discontinued issue tracker to their new one. Not only does this waste time, developers may also have issues porting from one issue tracker to another. For instance, Google Code has an indicator of severity, but GitHub does not, which may cause a loss of information (unless they put that information in another field, which has its own limitation, such as sorting). Another limitation of migrating from one issue tracker to another is that some fields that are functionally the same (like a summary) have different names or identifiers in issue trackers. An example of this is Google Code's "Summary" field, which may correspond to GitHub's "Title" and "Body" field<sup>1 2</sup>. Even though it may not require much time for someone to fix this small detail, as the project size and number of issues increases, so does total amount of developer time required.

Although one may think that it is not common for issue trackers to shut down, in the past two and a half years, both Google Code and Microsoft CodePlex have been shut down. Google Code was discontinued on January 25, 2016 and their tools for exporting no longer work<sup>3 4</sup>. Because of this, developers that want to grab data from Google Code will have to spend more time reading through Google's API to retrieve projects/issues or rely on unofficial and outdated third party scripts. This is a similar case for Microsoft CodePlex, which ended support on December 15, 2017<sup>5</sup>. Their official migration tool is now considered "historical and no longer works" just 4 months after Microsoft ended support<sup>6</sup>. The tools for these issue trackers are already dysfunctional, making it difficult for developers to port their projects and issues to other issue trackers.

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 an

---

<sup>1</sup> "Google Code Archive - Long-Term Storage for Google Code Project Hosting." *Google*, Google, [code.google.com/archive/schema](https://code.google.com/archive/schema).

<sup>2</sup> "GitHub REST API v3." *Issues | GitHub Developer Guide*, GitHub, [developer.github.com/v3/issues/](https://developer.github.com/v3/issues/).

<sup>3</sup> "Bidding Farewell to Google Code." *Google Open Source Blog*, [opensource.googleblog.com/2015/03/farewell-to-google-code.html](https://opensource.googleblog.com/2015/03/farewell-to-google-code.html).

<sup>4</sup> "Export to GitHub - Google Code." *Google*, Google, [code.google.com/export-to-github/](https://code.google.com/export-to-github/).

<sup>5</sup> Bright, Peter. "Microsoft Closing down CodePlex, Tells Devs to Move to GitHub." *Ars Technica*, Ars Technica, 31 Mar. 2017

<sup>6</sup> Microsoft. "Microsoft CodePlex WorkItem Migrator." *GitHub*, GitHub, [github.com/Microsoft/CodePlexWorkItemMigrator](https://github.com/Microsoft/CodePlexWorkItemMigrator).

easy-to-use tool to migrate issues from many issue trackers to one unified file format will be instrumental to this goal. Not only is this unified file format significant, but speed and efficiency will also play a big role in this project. 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. This will allow developers who are on the move without a network connection to still access issues, which increases productivity. Because Track Forever runs locally and is a free and open source software, a company can use this tool on their own servers and extend it as they please. This may prove to be instrumental to productivity and resilience as companies no longer need to rely on a service like GitHub, which has been recently targeted by distributed denial of service attacks<sup>7</sup>. Companies that rely on an online issue tracker like GitHub will have its productivity hampered during such incidents.

## 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 format, which will include a project file with issue files in JSON. For now the project file will include name, owner name, and description. The issue file will include fields shared by the most popular issue trackers we plan to support: summary, status, assignees, submitter name, comments (as another short structure), and time created/updated/closed.

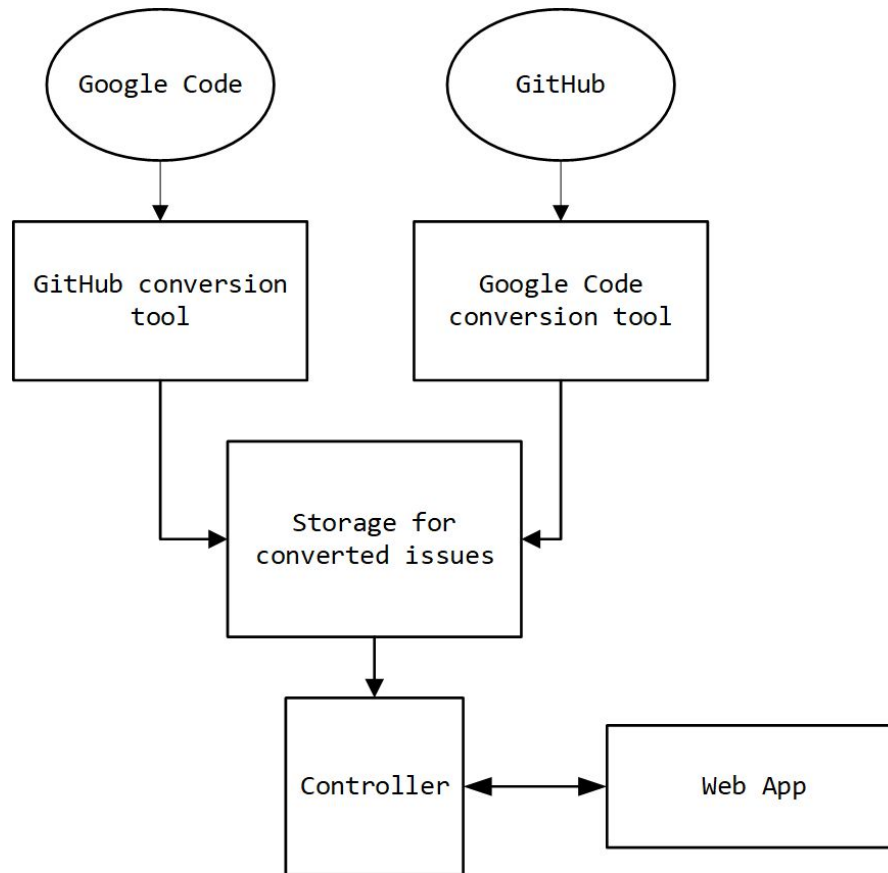
Although we initially plan to build our implementation to import issues from the Google Code Archive API, we will extend our program to include compatibility with popular issue trackers that have similar API support. For example, we could start with some of the several popular issue trackers like Jira 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.

In order to read our unified file format, we will create a webapp similar to GitHub's issue tracker to view issues offline (see fig. 2 below). The website will at first only support viewing, but could over time become its own usable issue tracker.

---

<sup>7</sup> "February 28th DDoS Incident Report." *GitHub Engineering*, GitHub, 1 Mar. 2018, [githubengineering.com/ddos-incident-report/](https://githubengineering.com/ddos-incident-report/).

## Architecture



(fig 1.) Architecture diagram

Track Forever will have the following architecture. There will be a conversion tool for each type of issue tracker. We'll begin by supporting Google Code and GitHub. Once converted, the formatted issues will be stored together. The web app will call the controller to fetch issues, as well as importing new issues.

## Technologies

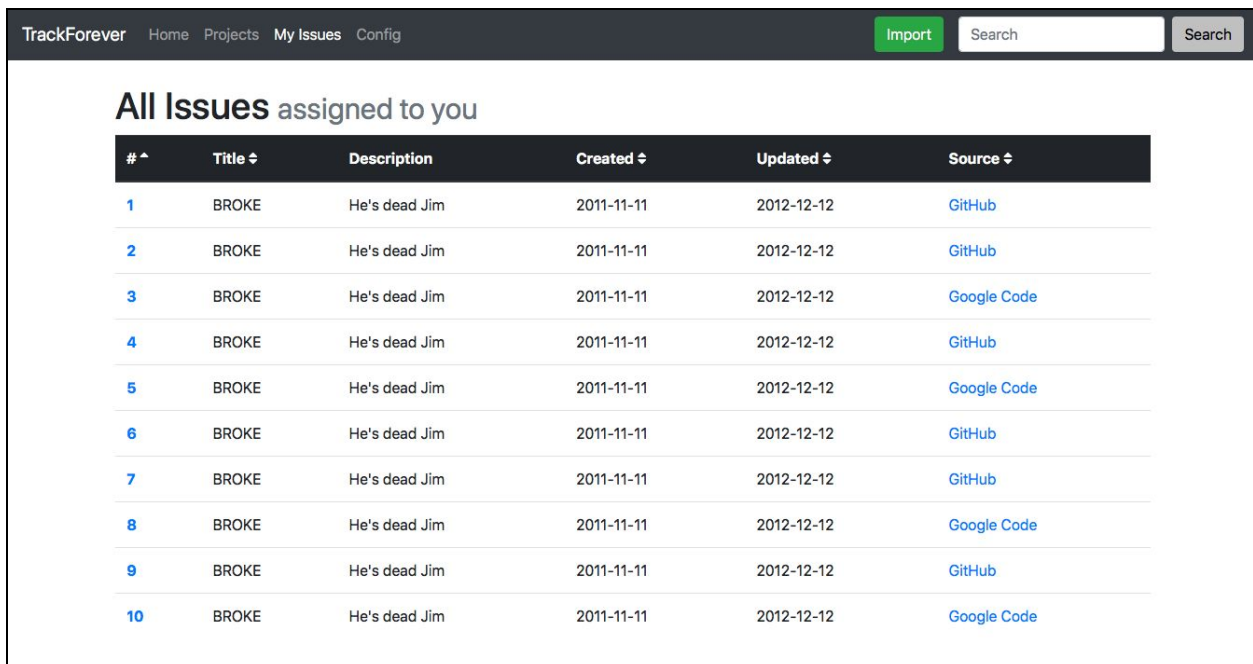
We chose JSON as our file format for issue tracker data because it is human-readable and has broad support across different languages. These are both qualities that will ease the development of plugins and clients from contributors outside our team.

As the issue trackers we plan to support are varied, the technologies we use to consume their API and convert to our format may be varied as well. We chose Kotlin for the conversions because it is easy to parse JSON in the language and many issue trackers including ours use JSON. However, it will be possible for our project to be extended for separate plugins using different technologies so

long as they produce issues in the same format. Once in this format, the issues will be added to a SQL database and served to the front-end through a RESTful API.

The front-end of Track Forever will use Angular 5, a web application framework developed by Google. Angular was chosen over other popular web frameworks (or no web framework) because it is easy to set up with many features including Karma for unit testing and tslint for code style integration with the IDE. Angular also uses Typescript by default, which will help with code readability and static verification. The web app will also use bootstrap css to speed up development and maintain a consistent style.

## Interface



# ^	Title ^	Description	Created ^	Updated ^	Source ^
1	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
2	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
3	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">Google Code</a>
4	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
5	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">Google Code</a>
6	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
7	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
8	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">Google Code</a>
9	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">GitHub</a>
10	BROKE	He's dead Jim	2011-11-11	2012-12-12	<a href="#">Google Code</a>

(fig. 2) Issue tracking website mockup

The webapp will be similar to GitHub's. Its main page will display a list of issues, allow filtering, and sorting. Clicking a specific issue will open a details page containing more information. There will be an import button on the top corner, which will open a new page and allow the user to import more issues.

## 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. What these trackers lack, however, is a comprehensive or standardized method to import issues from other trackers. Moreover, many

of these trackers 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<sup>8</sup>. 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<sup>9</sup>.

## Challenges and Risks

There is always the risk that an attempt at a “unified” standard becomes just another competing standard. 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 to build their own implementations of the issue viewer, encouraging participation and growth in the project.

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.

---

<sup>8</sup> Gu, H., L. Zhao, and C. Shu. *Analysis of Duplicate Issue Reports for Issue Tracking System*, 2011, SCOPUS, [www.scopus.com](http://www.scopus.com).

<sup>9</sup> 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>

## 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.

In order to provide more concrete evidence of our project's viability, we can demonstrate using well-known projects on public repositories. For example, we can look for projects on discontinued trackers, such as Google Code, and then demonstrate step-by-step how our project can easily export those issues Google Code into, say, Github. We could also create a dummy project on some modern issue tracker that uses a variety of features of that issue tracker, and then export those issues to another issue tracker. By demonstrating that our project can handle real-life and potentially complex usage cases, we can convince potential users that our project is worth their time.

## 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 (fields, representation on disk, etc.) for our issue tracker
- ✓ Design the backend system architecture
- ✓ Design a mockup of the UI for the web app
- Design API contracts for interfacing between web app and backend

**Week 4 - Implementation begins**

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

**Week 5 - Implementation continues**

- Finish a conversion tool for at least one more bug tracker
- Implement the REST API for interfacing between the webapp 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**

- Run tests and document 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 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
- Finish final presentation slides

**Week 11 - Final presentation and report**