BusyB

A Data Collection System
Designed for the Front Line Support Team
At Pixar Animation Studios

ABSTRACT

Pixar Animation Studios develops and maintains a custom 3D-graphics software tool (for the purposes of this paper the tool is called AnimationX) used for digitally animating all of their movies. With each new movie, new features and improvements are added to AnimationX to support new capabilities. For example, in the movie Brave, which is scheduled for release in June 2012, new software features were added to support realistic simulations of human hair movement. Since Pixar's movies primarily rely on AnimationX, it is critical that the tool perform quickly and without error. As a result, Pixar is highly invested in measuring and maintaining Animation X's software quality. Software quality is measured using a number of different metrics at Pixar. One metric is speed, a measurement of how fast or slow the software runs. AnimationX currently has the feature that it generates a log file everytime a new instance of AnimationX is opened. This log file, called a busylog, is stored on the same machine that opened the AnimationX program and it contains valuable information about the performance of that particular running instance of the software. Busylogs are of particular interest to Pixar's Front Line Support Team who use the logs to help diagnose slowness issues when slowness is experienced by Pixar artists using AnimationX. Before the BusyB project, these log files were distributed across the entire studio on individual user's machines with no system for tracking or archiving these logs. Furthermore, there did not exist a way to easily browse and view multiple log files. BusyB is the name of the data collection system that Iris Cheung designed and implemented for collecting, storing, and viewing information from busylog files in an efficient and user-friendly manner. BusyB was successfully deployed at Pixar in April 2012. As of the day it was deployed it had gathered approximately 50,000 logs across 1,000 live production machines across the studio. The project was well received in a presentation to senior technical management at Pixar. BusyB lays the structural foundations for gathering and sifting through busylog data. With this foundation in place, there is now greater potential for gaining insight from the busylog data to answer future questions such as: "How many people are experiencing the same slowness issue?" or "How did the slowness compare between one software release and another?".

TEAM

BusyB was designed and implemented at Pixar Animation Studios by Iris Cheung for her Master's thesis at the UC Berkeley School of Information. Iris received guidance and support from several Pixar employees. First and foremost, Ian Buono, managed Pixar's high-level expectations for the project and is responsible for garnering the support of senior management that ultimately paved the way for the existence of the project. Sowmya Natarajan managed legal considerations regarding releasing code and other digital artifacts outside of Pixar. Ethan Karson, Kitt Hirasaki and Jason Kim provided technical and design mentorship respectively throughout the project. Kitt coined the name 'BusyB' for the project. Richard Yoshida provided great feedback about the use cases that BusyB should aim to solve. Finally, Mckay Farley and Jason Williams helped review code and give advice regarding technical challenges encountered in the project.

ORIGINS

The birth of the BusyB project stems from the greater desire that the Studio Tools Quality Assurance (QA) department at Pixar has for conducting increasingly intelligent assessments of software quality. Long before it was decided that busylogs would be the focus of my Master's project, we conducted a couple brainstorming sessions to discover what kind of questions the Tools QA department would be most interested in answering. These brainstorming sessions covered potential uses for the busylog data as well as the potential uses for a second type of log, the crashlog.

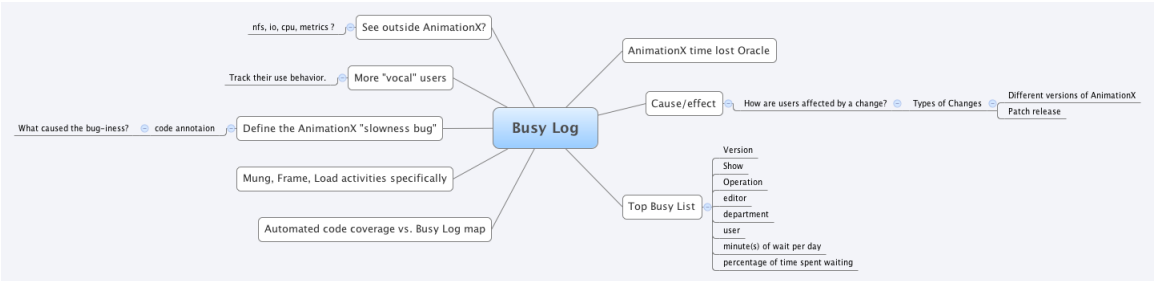

Figure 1. Brainstorm map of interesting attributes related to busylogs

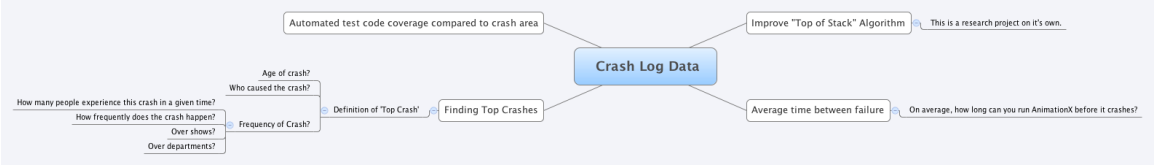

Figure 2. Brainstorm map of interesting attributes related to crashlog data

Similar to the busylog, the crashlog is stored on the user's local machine running an instance of AnimationX. The crashlog is different from the busylog in that it is only generated if AnimationX crashes, wheras a busylog is generated every time an instance of AnimationX is opened. The purpose of the crashlogs is to record stack trace information that occurred at the time of the crash so that such information can be used to diagnose the cause of the crash.

The following are some questions that could be asked of the busylog data

Using the busylog data:

- What are the top busy operations? In other words, which operations are the biggest offenders in terms of the portion of time an AnimationX user spends waiting for the operation to complete?
    - How many ways could we slice this ranked list and which way would be the most beneficial or illuminating? For example, the top busy operations might be different per version of AnimationX, per department (ex. Animation department versus Layout deparment), or per show (ex. Finding Nemo versus Monster's Inc.).
        - Slicing the top busy operations according to user might be particularly interesting because it could illuminate the validity of a user's complaints about particular slowness issues. In other words, user John Doe might complain that operation Z (ex. constraining a prop to a model) occupies a majority of his wait time when using AnimationX, when in fact his perception of the wait time is inflated and not corroborated by the percentage of wait time indicated by the top busy operations list.
- How can we compare the "area" of automated test coverage with the "area" of busylog coverage? It would be interesting to develop a way to visualize how the two spaces overlap. In this case, areas of overlap would represent sections of AnimationX code that have both automated test result data as well as busylog data describing its performance. A knowledge of where these overlaps exist could be useful to a test engineer because the engineer could then, in the areas of overlap, use the busylog data to help inform whether or not the automated test is conducting it's test accurately and precisely.

Using the crashlog data:

- What is the mean time between crashes for a given running instance of AnimationX? In other words, how long can an animator expect to run a single instance of AnimationX before it crashes?
- Similar to the busylogs, could we compare the "area" of automated test coverate with the "area" of code which commonly causes crashes to occur?
    - The current method of determining the cause of a crash involves algorithmically determining the most relevant piece of information from the stack trace. This piece of information is called the "Top of Stack". So in order to answer the preceeding question above, it would be necessary to first look into improving the "Top of Stack" algorithm, which, after consulting Shiqi Yang, a test engineer on my team at Pixar, is a subject that could be an entire Master's project on its own.

These are only a few highlights from a longer list of questions generated from these brainstorming sessions. I ultimately decided to focus my Master's project on the busylogs since, at that time, there existed no infrastructure for collecting the busylogs and storing them in a centralized location, and consequently no infrastructure for doing any analysis over the entire collection busylogs; this is in contrast to the crashlogs, which already has an automated system in place for the storage and analysis of various crashlog attributes (ex. the "Top of Stack" algorithm).

I decided that the goal of my Master's project would be to build a system that
1) Lays the foundations for building a corpus of busylogs collected across the studio.
2) Makes searching and viewing busylogs quick and easy.

USE CASES

To ensure that the system I was designing would meet real needs within the Studio Tools QA department, I first gathered use cases from members of the Front Line Support (FLS) team. Specifically I received feedback from Ethan Karson, Richard Yoshida, and Kitt Hirasaki. Their feedback can be summarized into the two general use cases below:

Use Case 1: Determining the cause of AnimationX slowness.
When a user, or group of users, contacts Front Line Support (FLS) because they are experiencing slowness, FLS uses the busylogs as one form of data to help determine what is causing the slowness. Sometimes users experience a single event of slowness when a particular operation is performed (ex. making a take). Other times users are experiencing intermittent slowness (ex. calls to the perforce software revisioning client).

Use Case 2: Assessing which operations are slow and who is experiencing slowness.
It would be nice to have quantifiable data to support which operations in AnimationX are the "biggest offenders" with regard to slowness. It would also be nice to be able to easily compare which populations (ex. animators vs. layout artists) are experiencing more or less slowness as well as what kind of slowness. It would also be interesting to compare populations according to which show the populations belong to, for instance, "which show is experiencing the most slowness? And why?"

PROTOTYPES

Recall that there are two goals I set out to achieve for my Master's project:

1) Lay the foundations for building a corpus of busylogs collected across the studio.
2) Makes searching and viewing busylogs quick and easy.

The second goal, would require an application with a user-interface to allow the user to search for and view busylog data. At this stage in the project, there were still several unresolved issues about the nature of this "viewer" application. For instance, the major questions included:

- What set of features are the most critical for incorporating into the application?
- Should the application be a web application or a stand-alone, PyQt, Linux application?
- How should the project be scoped such that it can be fully completed within a semester time-frame?

These questions and more were discussed over several meetings with Ian Buono, Ethan Karson, and Kitt Hirasaki. Deciding on a list of critical features was difficult, partly because it was dependent upon the personal workflow of each Front Line Support member. For example, a member that is concerned more about preempting slowness issues might value an application that presents alerts when users are experiencing uncustomary slowness, whereas a member that is frequently contacted by AnimationX users experiencing slowness might value an application that is more centered on fast and efficient searching through busylogs to help aid the diagnostic process. Additionally, we considered whether this application would have audiences outside of the Front Line support group, such as technical management or show specific support personnel.

To facilitate these discussions, I created two paper prototypes of the "viewer" application which incorporated many of the ideas generated during the discussions.

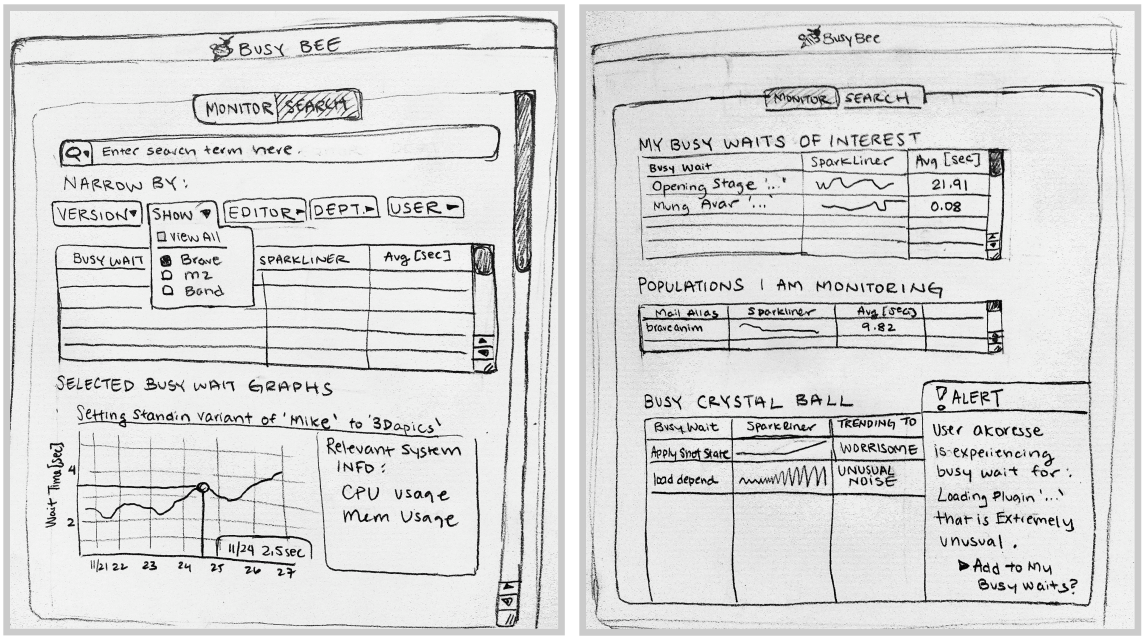

Figure 3a and 3b. Paper prototypes of Busyb

It was clear from these discussions that there was plenty of potential for interesting data analysis and visualization. The limiting factor would be time. Considering this limiting factor, some high-level goals for the project emerged:

- The application created should first and foremost be production quality code. (ex. undergo peer reviews, use systems approved storage locations, checked into mainline branch of code repository) In other words, even though this code is being created for an academic project, the intention is not for this application to be a "prototype". It will in fact be a fully-functional and deployed addition to the mainline repository of code.
- Related to the point above about production quality, it is more important for the end-product be functional than for the end-product to have several half-finished features.
- Use Case 1 is more important than Use Case 2. In other words, it is more important to get a solid infrastructure in place for collecting logs and viewing them, than it is to perform analytics on the busylog data.

Ultimately we decided that my course of action would be to design a solid infrastructure required for the collection and viewing of the corpus of busylogs, and any leftover time could be devoted to either a second design iteration or an implementation of a "blue sky" feature.

SYSTEM DESIGN & IMPLEMENTATION

The system I designed comprises of three parts:

- A file crawler.
- A database.
- A file browser application, called the Busy Viewer.

File Crawler

This file crawler is a Python script used for finding busylog files on local user machines, storing them in an archive location on hard disk, and recording an entry about the stored busylog in a database. This script comprises of five separate files:

busybCrawer.py
This is the top-level script used for running the file crawler.

busyLog.py

This is a class, named ‘BusyLog’, used for modeling the attributes of a busylog. Such attributes include the session start date for the log, the last modified date, the date retrieved, the path to the log’s archive location, the path to log on the user’s local computer, the machine name storing the log and the username of the user that generated the log.

crawler.py

This is a class, named ‘Crawler’, used for crawling over machines for the users specified in the xml file supplied to busybCrawler.py. The Crawler class has methods for getting the date of the newest log stored in the database for a particular user, finding the latest logs stored on a particular user’s machine, archiving logs, and storing an entry in the database about an archived log.

userHandler.py

This is a class, named UserHandler, which is used for parsing the xml file, passed into busybCrawler.py, which specifies which AnimationX users to find busylog files for.

The reason the script is broken up into five separate files is because it is a Pixar coding standard to have one Python class per file. Conforming to this standard makes the code more familiar to navigate for other Pixar engineers. This is one example of many coding standards I had to adhere to for my code to satisfy production-quality standards.

The script finds busylogs on machines belonging to the users specified by an xml file, named busybUsers.xml, that is passed into this script using a Python option parser. Figure 4 shows an example of a valid busybUsers.xml file. The xml schema is very simple. The schema includes two types of elements: ‘user’ and ‘alias’. A user element expects the value of a Pixar username. The alias element expects a Pixar mail alias. The reason we decided to include mail aliases as a way to specify a group of users is because various groups of Pixar employees are already conveniently grouped by mail aliases. It is natural for a Front Line Support engineer to be working with a group of Pixar employees that all belong to the same mail alias. This xml file is meant to be easily editable by a Front Line Support engineer so that he/she can specify exactly which users to collect busylogs for.

```
<busybinput>
    <user>irischeung</user>
    <user>ekarson</user>
    <user>ibuono</user>
    <alias>menv30-users</alias>
    <alias>busyb</alias>
</busybinput>
```

Figure 4. Example of a valid busybUsers.xml file

Busylogs are collected only from machines that are awake (ie pinged recently), and running Linux, this is because AnimationX only runs on Linux machines. Also, it is uncessary to look for logs on machines which are sleeping and are, in other words, not being used currently.

A busylog is 'archived' by being copied over to an archive location. The location in use, as of March 2012, which has been allocated by the systems department at Pixar, is /depts/tools/busylogs/. An alternative archive location can be specified by passing an option into this script using the option parser.

Beneath the archive location, busylogs are stored using the following directory structure: /username/machine/<the user's TMPDIR>/YYYY/MM, where YYYY/MM are the year and month of the session start date for the busylog.

Currently, the value of the user's TMPDIR is hard coded in the class 'Crawler'. A way of determining the value of a user's TMPDIR environment variable without logging as the user him/herself has yet to be discovered.

In addition to being stored in the archive location, “metadata” about each busylog is recorded in the 'busylogs' table in the 'busyb' database to keep track of which logs have been archived.

Database

The busyb database is a MySQL database hosted on an internal server at Pixar. The busyb database contains a single table, named ‘busylogs’, used for storing attributes about each busylog that is copied over to the archive location. Such attributes includes:

- session_start_date: A timestamp of when the AnimationX session began for this busylog.
- last_modified_date: A timestamp of when the busylog was last modified.
- date_retrieved: A timestamp of when the busylog was last retrieved by the file crawler.
- busylog_archive_path: The path to the busylog stored in the archive location.
- busylog_local_path: The path to the original busylog stored on the user’s local machine.
- host: The name of the machine that created the busylog.
- user: The name of the user that created the busylog.

COLUMN NAME	DATA TYPE
session_start_date	datetime
last_modified_date	datetime
date_retrieved	datetime
busylog_archive_path	string(64)
busylog_local_path	string(64)
host	string(32)
user	string(32)

Figure 5. The busylogs table in the busyb database

Busy Viewer

The Busy Viewer is the graphical user interface (GUI) used for viewing the busylogs which have been stored in the archive location. We decided to create a stand-alone, PyQt, Linux application over a web application because there is more internal expertise and motivation within Pixar for maintaining and extending stand-alone applications than there is for web applications.

Upon starting the viewer the user is greeted with a relatively blank application and two search fields, one for user(s) and one for a mail alias.

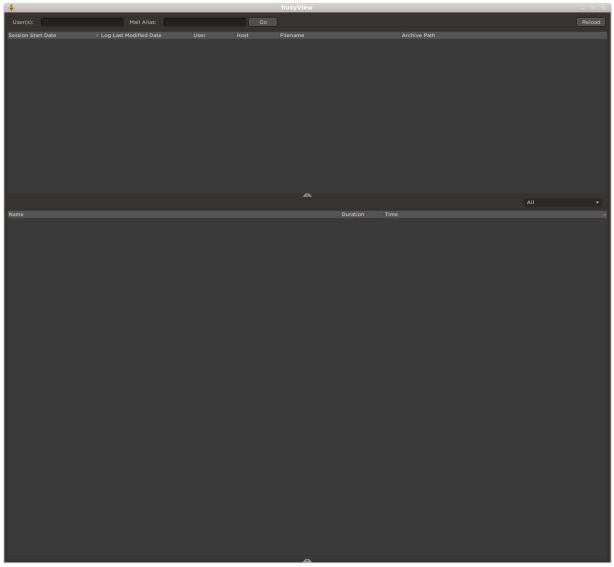

Figure 5. Busy Viewer on start-up

A user can enter multiple complete or partial usernames, separated by commas, into the user(s) search field, and can also enter a single mail alias into the mail alias search field. Pressing enter or go will then populate the first table with all of the logs found for those users and mail alias specified.

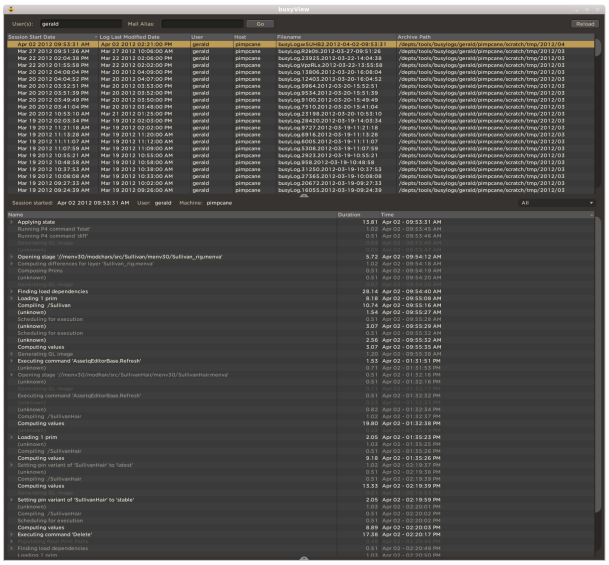

Figure 6. Busy Viewer with a username entered in the search field

Clicking on a particular log file in the table brings up the individual log file for viewing in the table used for viewing individual logs below. Clicking a particular busylog operation, also known as a busy scope, in the individual log file view will then highlight when that particular operation took place in a stacked-timeline visualization at the bottom of the application.

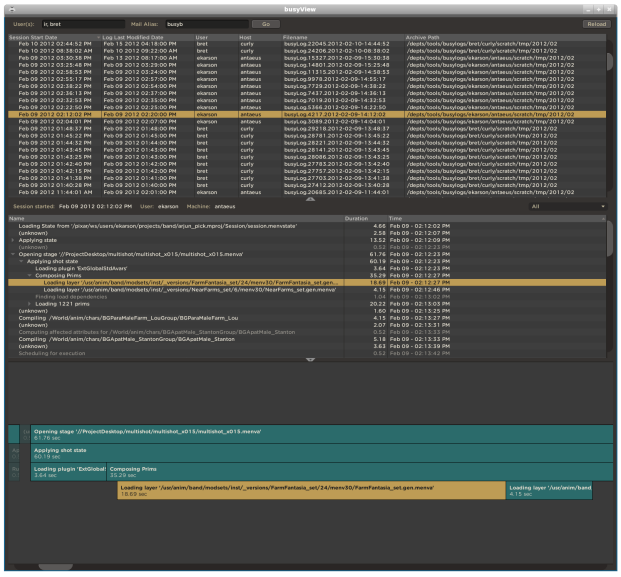

Figure 5. Busy Viewer with a log selected and a busy scope selected

RESULTS

BusyB was successfully deployed at Pixar in April 2012. As of the day it was deployed it had gathered approximately 43,000 logs across 1,000 live production machines across the studio. There are, however, an estimated 250,000 logs currently stored on local machines across the studio, with a rate of approx 1,000 logs generated per day. The file crawler script takes approximately half a second to store each log, thus it would take approximately 6 full days of running in order to gather the total quarter million logs that exist at the studio. A combination of technical challenges and limited time prevented the script from gathering all of the existing logs. One technical challenge in particular was in trying to diagnose why the file crawler script would hang occasionally when run. An early iteration of the file crawler used the opensource python ssh module paramiko, which proved to be the cause of some instances of hanging. This module was later removed and replaced with

calls to Python's subprocess module. In spite of this fix, as of April 3 (the Pixar prescribed end date for the project), it was observed that the script would still, on occasion, hang when run. This symptom leaves the file crawler script in functional, yet buggy, state, and is what has prevented the script from being hooked up to a cron utility for running periodically.

CONCLUSION

Over the course of the semester, the singular overarching goal was to create and deploy a live system that would improve the lives of FLS team members in terms of access to busylog data. This goal was achieved in the most fundamental sense through the creation of the file browser, database, and busylog viewer- all of which are running live and in a state which is ready to be used by FLS team members today. BusyB has received positive feedback from Ethan Karson and Richard Yoshida, primary members of the FLS team, in regards to an enhanced capability for finding, browsing through, and viewing busylog files.

FUTURE PLANS

There are many potential ideas for improving and extending upon the BusyB system:

Correlation with other software metric logs

AnimationX has other log files which gather different software metrics and it might be useful for BusyB to read in these other logs to present more data to the FLS member which may help the diagnosis for why a user is experiencing slowness issues.

Storing busylog data at a finer granularity

Storing individual busy scopes (i.e. the individual operations) within each busylog in a database and developing the scalable infrastructure to mine and parse each individual operation would open doors for gaining insight on which particular operations are the "biggest offenders" in terms of causing software slowness.

Notification versus exploration

The current BusyB system provides the most fundamental functionallity for FLS members and was designed to fulfill their most basic and current need, which was to search for a busylog given they have received a message from a user or group of users experiencing slowness. Now that this need has been met, it would be exciting to add on features to BusyB that provide analysis over the log files that have been collected. Such analysis could be the basis of features which provide "smart" notifications to FLS members to give FLS members a heads-up before they even receive complaints from users.

Exciting challenges for data visualization and user experience

The busylog viewer in its current state fulfills the FLS team's most basic needs. Given that BusyB will be extended to perform analysis and generate statistics about slowness, there is a lot of exciting potential for designing how such statistcs will be presented to an FLS member to achieve the most efficient and intuitive user experience. Often an FLS member will be assisting several different artists from any number of different departments. Designing data visualizations and a user experience to cater to an FLS members workflow would be a challenge that could inspire new ways of displaying information.

CODE

The code developed for this project is owned by Pixar Animation Studios and is available upon on request. Depending on the intended purpose for viewing the code, the person requesting may be required to sign a non-disclosure agreement.

SPECIAL THANKS

Kimiko Ryokai, Ian Buono, Sowmya Natarajan, Kitt Hirasaki, Ethan Karson, Richard Yoshida, Jason Kim, Mckay Farley, Jason Williams