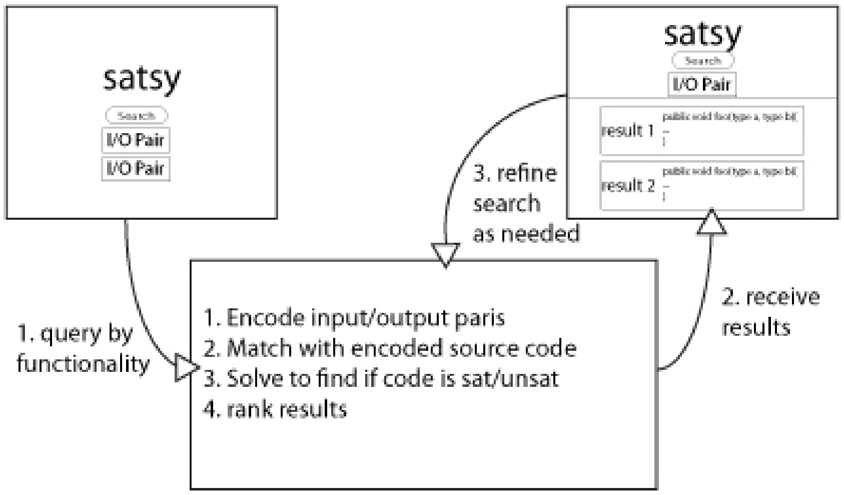
Satsy Project Plan 2

Team: May14_13
Date: November 15, 2013
Members: Carl Chapman, Cody Hoover, Cole Groff, Kaitlin McAbee, Trevor Lund
Advisor: Kathryn Stolee

Problem Statement

The team will build a search engine that will search for source code based on functionality. The concept for this code search engine was originally developed by Kathryn Stolee [1]. She has named the project ‘satsy’. To further develop satsy, the team will create an online interface where users can enter input-output pairs, and develop a method of parallelizing the search process, and design an algorithm for ranking results. This project will incorporate existing modules created by Dr. Stolee that encode input-output pairs into a format that can be fed into an satisfiability modulo theory (smt) solver, and a repository of encoded code blocks. It will also use Microsoft’s Z3 smt solver [2].

Concept Sketch

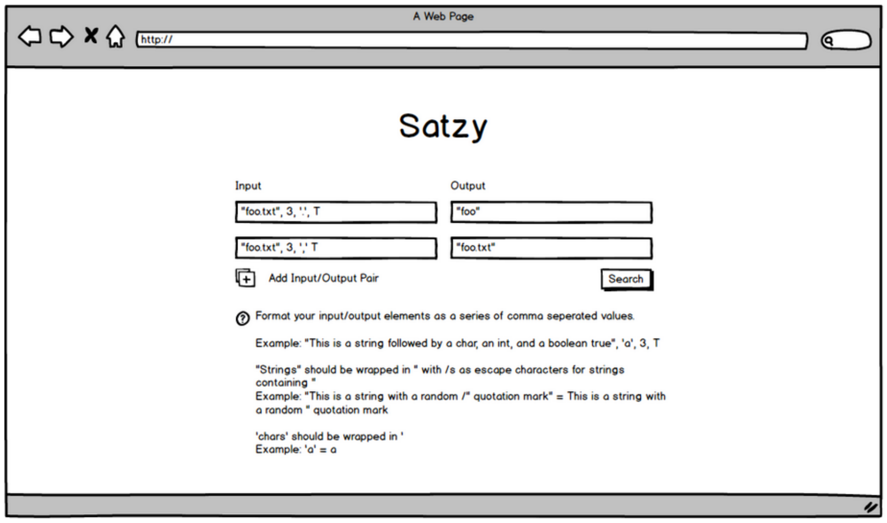

satsy
Search
I/O Pair
I/O Pair
satsy
Search
I/O Pair
result 1
result 2
3. refine search as needed
1. query by functionality
2. receive results
1. Encode input/output paris
2. Match with encoded source code
3. Solve to find if code is sat/unsat
4. rank results

As an example of how satsy is intended to be used, a user may specify the input “foo.txt” and the output “foo” as one input-output pair (optionally entering more input-output pairs), and then receive a list of code blocks that come closest to satisfying this behavior.

User interface description

The interface will be a website with a single dynamic page that changes from an initial blank search page format to a format that displays results and allows more searches.

1. Initial Search Format: users will enter input/output strings representative of the desired behavior. When the user hits search, the page will change to the second format.



"foo.txt", 3, '.', T | "foo"

"foo.txt", 3, ',' T | "foo.txt"

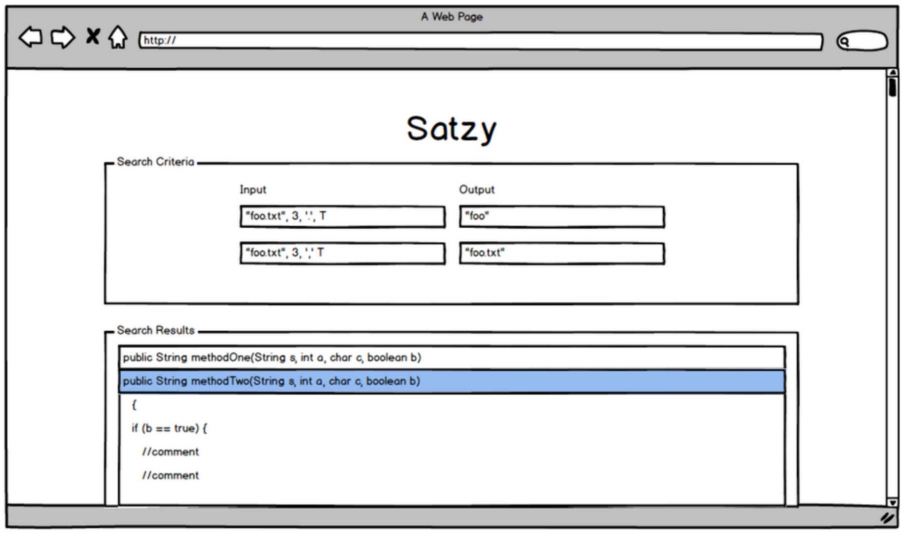
Format your input/output elements as a series of comma seperated values.

Example: "This is a string followed by a char, an int, and a boolean true", 'a', 3, T

"Strings" should be wrapped in " with /s as escape characters for strings containing "
Example: "This is a string with a random /" quotation mark" = This is a string with a random " quotation mark

'chars' should be wrapped in '
Example: 'a' = a

2. Results And Refine Search Format: the users query will remain visible and modifiable, but will be partially hidden for searches with more than two input-output pairs. A result section will become visible and populated with top-ranked results. Each result will contain the source code of the given function, and information about what input-output pairs matched it. Users can expand the input-output area and refine their search as needed.



"foo.txt", 3, '.', T | "foo"

"foo.txt", 3, ',' T | "foo.txt"

```
public String methodOne(String s, int a, char c, boolean b)
public String methodTwo(String s, int a, char c, boolean b)
 {
 if (b == true) {
   //comment
   //comment
```

Functional requirements

- Satsy will be able to accept multiple input-output pairs as search terms from each user, up to 10 pairs.
- Satsy will be accessible from anywhere that has internet access.
- Satsy will return a set of source code snippets given a set of input-output pairs.
- The source code that Satsy returns should be able to take the given inputs as parameters and produce the given output as a returned result.
- The source code results nearer to the top of the search result page should be satisfiable by most or all of the input-output pairs.
- Source code results that are satisfiable by all of the input-output pairs will be sorted by other criteria to be specified later.
- Satsy needs to provide a clean interface to interact with the user, to be determined by user feedback.
- Satsy will provide feedback to the user if they didn't format their search terms correctly or if an error occurred.
- Satsy will display an input search box and an output search box.
- Users will be able to add additional sets of search boxes on the search page.
- The system will handle Java string, int, boolean, and character inputs, and allow extensions for further language support.

Non-functional requirements

- Users should quickly be shown the results of their search upon submitting their search criteria.
    - Metric: There shall be no more than 5 seconds between the user clicking the search button and the user seeing their search results.
- Searching with Satsy should be intuitive to the user
    - Metric: Less than 15% of users should express confusion when using Satsy.
    - Metric: 90% of users should be able to construct a valid search query within 1 minute of using Satsy.
- Satsy needs to support returning many different source code results.
    - Metric: Up to 1000 source code results can be returned.
- Satsy will be able to handle multiple user searches at once.
    - Metric: Up to 100 searches at once will be supported.
- Results should be accurate.
    - Metric: Results should return with a false positive rate of <15% and a false negative rate of <5%.

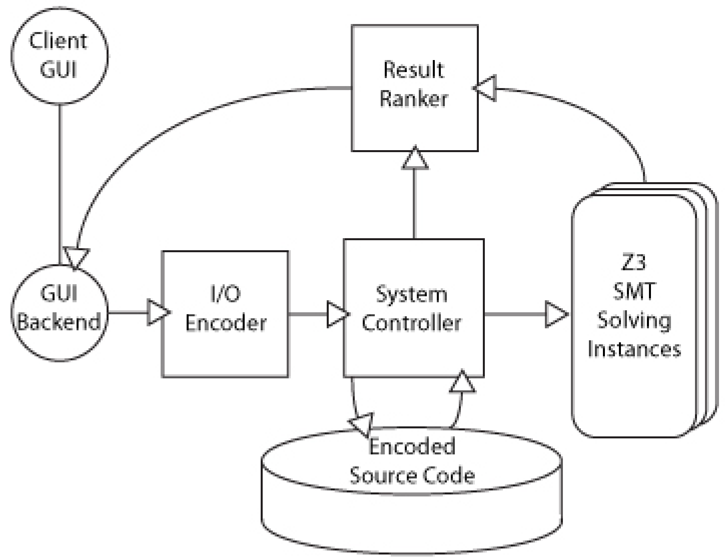
Market and literature survey

According to a survey done in 2011, most programmers, when faced with the task of searching for code, turn to Google [7]. By doing this, the programmer is given millions of search results. The usefulness of the top ten or so results depends on the level of experience of the programmer and the keywords used in the search parameters. This is where the satsy system comes in. Instead of searching by a description of the functionality of the code the programmer is looking for, they can search by the exact behavior based on the inputs and outputs of the code. In the research done by Dr. Stolee and her team at the University of Nebraska, they searched for the functionality of 17 code snippets using both Google and an early form of satsy. On average, Google returned 48.4 possible results and satsy returned 20.5 possible results. Of those results, Google had, on average, 1.5 results within the top 10 results on the page that actually produced the code that would act in the wanted manner. Satsy on the other hand, had an average of 8.5 with each search being capped at the maximum of 10 matching results. Overall, while Google produced more search results, satsy returned more accurate solutions within the top ten search results [6].

Deliverables

- A working system that parallelizes the use of Z3 to solve encoded input on multiple processors to increase operation speed and scalability.
- An attractive and simple user interface to be run in a web browser utilizing JavaScript, JavaScript libraries, and Java EE frameworks for Ajax and interface modification.
    - The first user interface deliverable will be constrained so the user will only be allowed to submit, for example, one input/output string.
    - A later iteration of the user interface deliverable will allow the user to modify the amount of input/output pairs they submit and change the type of submission pairs (string, integer, etc.).
- A ranking system for returning search results in a logical manner. Higher results will return code that more correctly satisfy the input/output pairs than lower results.

System block diagram



Internally, the system will follow the following sequence of events:

1. User enters I/O pairs into the ‘Client GUI’, transmitted to server via HTTP.
2. ‘GUI Backend’ passes input into the ‘I/O Encoder’ and waits for results.
3. ‘I/O encoder’ translates input-output pairs into smt-lib2 [3] format.
4. ‘System Controller’ fetches ‘Encoded Source Code’ that matches the signature of the input-output pairs.
5. ‘System Controller’ launches threads that use Z3 to determine if a given encoded source code block satisfies the input/output pair.
6. Results are captured and ranked. It is expected that some Z3 instances will take longer than the desired time period (~1 sec), and so will return uncertain results.
7. Ranked Results are returned to the ‘GUI Backend’ and sent to the user via HTTP.

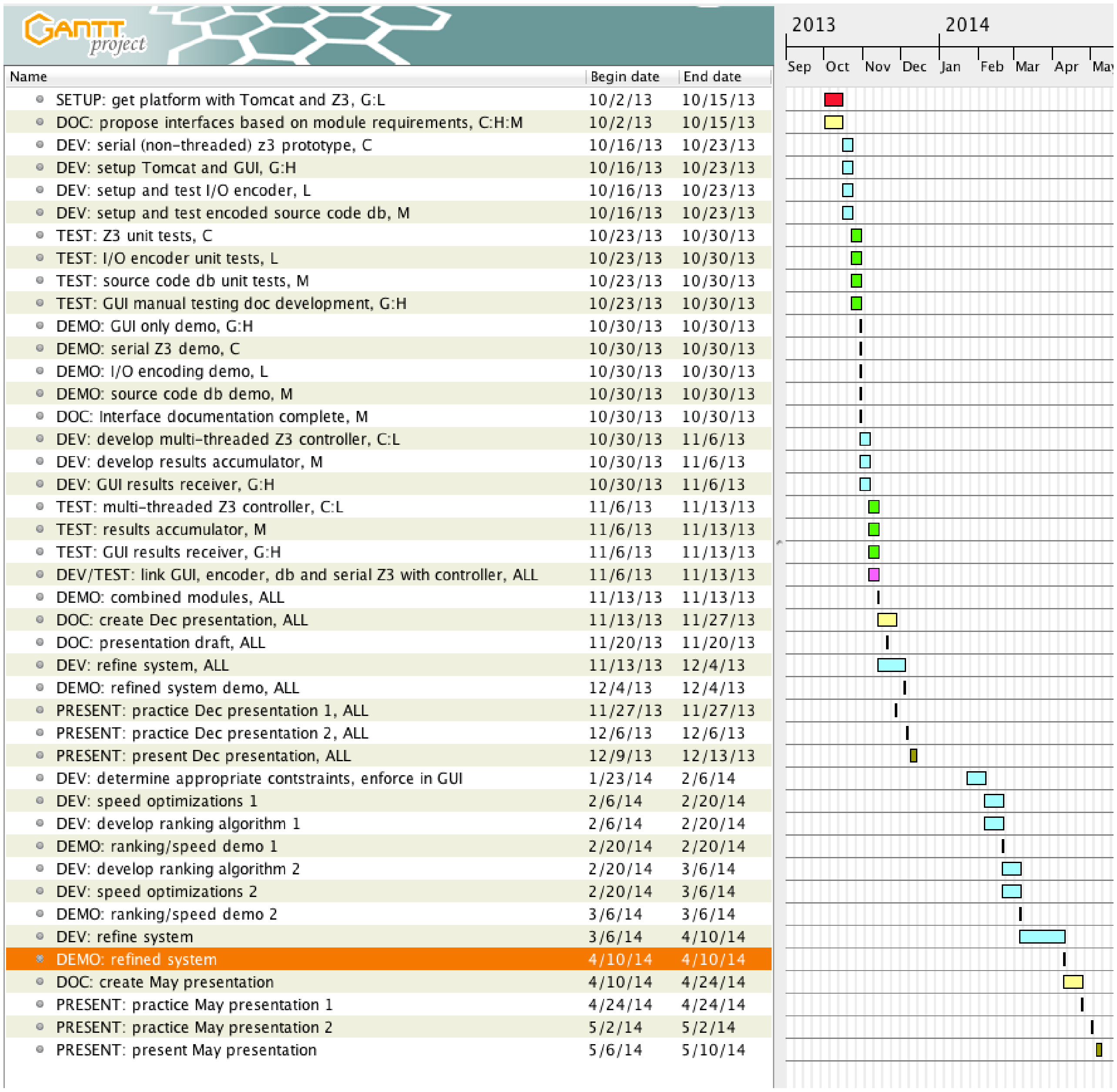
Operating environment

Choosing the right operating environment for our project is an important design decision that requires some investigation. The environment must support a functioning web server and the ability to launch multiple separate Z3 instances. One important consideration is that, ideally, the system would be able to scale to searching an arbitrarily large database of encoded methods while guaranteeing a specific response time. To keep costs down during development the database may be extremely small, but the vision of this project is to search a much larger set of encoded code blocks.

The website and back end code will run on Amazon's Elastic Cloud Compute (EC2) service. EC2 provides access to a virtual linux environment. Within this environment, a Tomcat-based server will instantiate servlets running the 'I/O Encoder', 'System Controller' and 'Result Ranker' modules in a Java 1.6 JVM.

Each Servlet serving a single user will manage an Executor service, launching one thread for each encoded code block-I/O pair combination. The project will use a MySQL 5.6.14 database stored on our EC2 server to store the encoded methods and source code. The user interface will be developed using the Bootstrap and Backbone Javascript libraries. A git repository on Bitbucket holds version-controlled files [10], and all documentation is stored in a Google Doc shared folder.

Work Breakdown Structure



Name	Begin date	End date
SETUP: get platform with Tomcat and Z3, G:L	10/2/13	10/15/13
DOC: propose interfaces based on module requirements, C:H:M	10/2/13	10/15/13
DEV: serial (non-threaded) z3 prototype, C	10/16/13	10/23/13
DEV: setup Tomcat and GUI, G:H	10/16/13	10/23/13
DEV: setup and test I/O encoder, L	10/16/13	10/23/13
DEV: setup and test encoded source code db, M	10/16/13	10/23/13
TEST: Z3 unit tests, C	10/23/13	10/30/13
TEST: I/O encoder unit tests, L	10/23/13	10/30/13
TEST: source code db unit tests, M	10/23/13	10/30/13
TEST: GUI manual testing doc development, G:H	10/23/13	10/30/13
DEMO: GUI only demo, G:H	10/30/13	10/30/13
DEMO: serial Z3 demo, C	10/30/13	10/30/13
DEMO: I/O encoding demo, L	10/30/13	10/30/13
DEMO: source code db demo, M	10/30/13	10/30/13
DOC: Interface documentation complete, M	10/30/13	10/30/13
DEV: develop multi-threaded Z3 controller, C:L	10/30/13	11/6/13
DEV: develop results accumulator, M	10/30/13	11/6/13
DEV: GUI results receiver, G:H	10/30/13	11/6/13
TEST: multi-threaded Z3 controller, C:L	11/6/13	11/13/13
TEST: results accumulator, M	11/6/13	11/13/13
TEST: GUI results receiver, G:H	11/6/13	11/13/13
DEV/TEST: link GUI, encoder, db and serial Z3 with controller, ALL	11/6/13	11/13/13
DEMO: combined modules, ALL	11/13/13	11/13/13
DOC: create Dec presentation, ALL	11/13/13	11/27/13
DOC: presentation draft, ALL	11/20/13	11/20/13
DEV: refine system, ALL	11/13/13	12/4/13
DEMO: refined system demo, ALL	12/4/13	12/4/13
PRESENT: practice Dec presentation 1, ALL	11/27/13	11/27/13
PRESENT: practice Dec presentation 2, ALL	12/6/13	12/6/13
PRESENT: present Dec presentation, ALL	12/9/13	12/13/13
DEV: determine appropriate contstraints, enforce in GUI	1/23/14	2/6/14
DEV: speed optimizations 1	2/6/14	2/20/14
DEV: develop ranking algorithm 1	2/6/14	2/20/14
DEMO: ranking/speed demo 1	2/20/14	2/20/14
DEV: develop ranking algorithm 2	2/20/14	3/6/14
DEV: speed optimizations 2	2/20/14	3/6/14
DEMO: ranking/speed demo 2	3/6/14	3/6/14
DEV: refine system	3/6/14	4/10/14
DEMO: refined system	4/10/14	4/10/14
DOC: create May presentation	4/10/14	4/24/14
PRESENT: practice May presentation 1	4/24/14	4/24/14
PRESENT: practice May presentation 2	5/2/14	5/2/14
PRESENT: present May presentation	5/6/14	5/10/14

On the Friday before Thanksgiving break (Nov. 22), the team will present a demo of a working system, live on the EC2 server. Near the end of the semester, the team will be refining the prototype and developing a presentation for dead week. In 2014, the team will begin work on the ranking algorithm design and in boosting performance, continuing with the pattern of developing towards a demo, presenting what has been done so far, getting feedback and returning to development. This process will follow the principles of the Agile development manifesto[5].

References

[1] Senior Design Project Proposal From, MAY14_13 - Stolee
[2] http://z3.codeplex.com/f
[3] http://www.smtlib.org/
[4] http://research.microsoft.com/en-us/um/people/leonardo/parallel_z3.pdf
[5] http://agilemanifesto.org/principles.html
[6] http://cse-apps.unl.edu/facdb/publications/TR-UNL-CSE-2012-0012.pdf
[7] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How well do search engines support code retrieval on the web? ACM Trans. Softw. Eng. Methodol., 21(1):4:1–4:25, Dec. 2011.
[8] https://bitbucket.org/carlchapman/satsy