Forager : Project Analysis Report

Cycle 2
November 29, 2012

by

Group4

Matthew Powell
Robin Mays
Thomas Couture
Samuel Hall

A project report submitted for
SWE3613 Software Engineering Systems
Fall 2012

Department of Computer Science and Software Engineering
Southern Polytechnic State University
Marietta, Georgia

Table of Contents

LIST OF ABBREVIATIONS

HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
OS	Operating System
PID	Process ID
SQL	Structured Query Language
UI	User Interface

1. INTRODUCTION

1.1 EXECUTIVE SUMMARY

Group4 is the producer of the website analysis tool Forager. Forager will allow a systems administrator or webmaster to easily scan their site for broken links and missing resources, and offer an easy way to generate and compare reports.

Forager is user friendly and portable. Users are provided access to reports, scanning, and sorting tools through a website produced using common web standards. This means that Forager is accessible from any PC, laptop, tablet, or mobile device regardless of the client OS.

1.2 PROJECT GOALS

Forager is a web based website analysis tool. A user will login to the service and be able to start a variety of scans of their website. When the scans have completed, they will then be able to examine the results of the scans and compare them in various ways.

Forager will allow scans to be generated starting from the front page, limited to specific subdomains, or limited by time and distance from the front page. It will also allow the user to use a list of broken links from a previous scan instead of revisiting the entire website.

Forager will then allow users to sort their scans based on page load time, response time, errors, and the individual subdomains on which the page was found.

Group 4 plans to complete Forager's features in two sprints. The project is expected to be complete by November 27, 2012.

1.3 CYCLE GOALS

Group4's goal is to complete Forager in two sprint cycles as outlined below.

Sprint 1

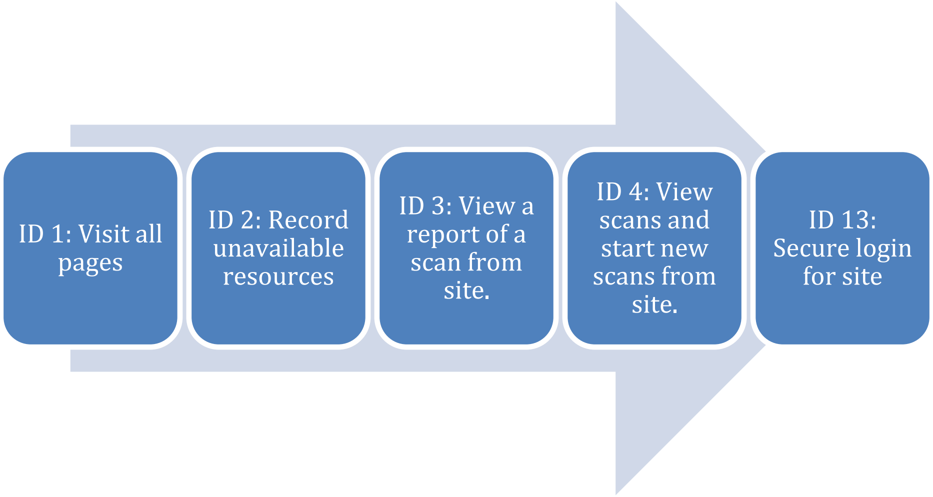

ID 1: Visit all pages
ID 2: Record unavailable resources
ID 3: View a report of a scan from site.
ID 4: View scans and start new scans from site.
ID 13: Secure login for site

Figure 1 User stories to complete for Sprint 1

Sprint 2

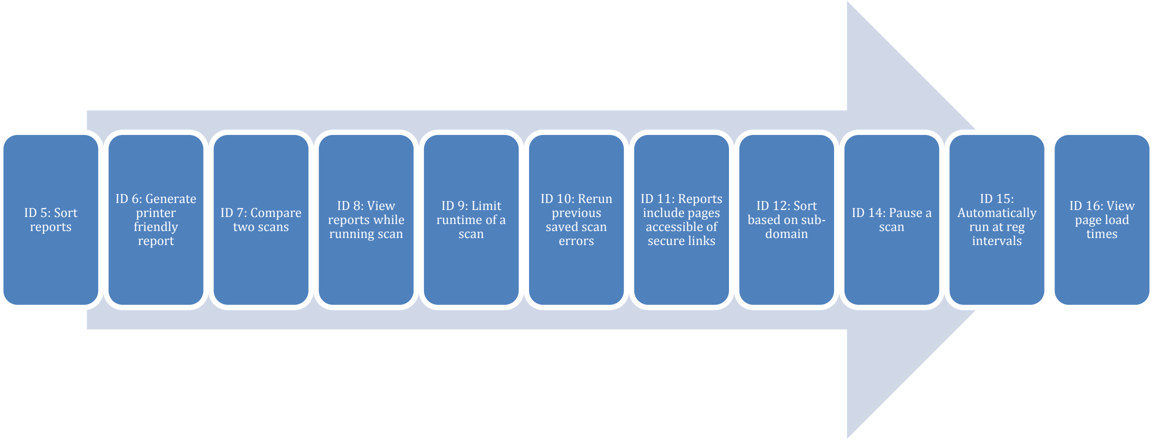

ID 5: Sort reports
ID 6: Generate printer friendly report
ID 7: Compare two scans
ID 8: View reports while running scan
ID 9: Limit runtime of a scan
ID 10: Rerun previous saved scan errors
ID 11: Reports include pages accessible of secure links
ID 12: Sort based on sub-domain
ID 14: Pause a scan
ID 15: Automatically run at reg intervals
ID 16: View page load times

Figure 2 User stories to complete for Sprint 2

2. REQUIREMENTS

2.1 PROJECT ENVIRONMENT, TECHNOLOGY, HARDWARE, ETC.

Forager is a web application written in a combination of PHP5 and Python. PHP is a widely-used, general-purpose scripting language that is especially suited for Web development and can be embedded into HTML, the primary markup language for displaying web pages in a web browser. Python is a common scripting language that has a rich set of features for interacting with webservers and processing HTML data. Both the Python and PHP components communicate with a PostgreSQL 8.4 database back end and the user interface is served by the Apache 2.2 webserver. This selection of mature, multi-platform technologies will allow Forager to run with little modification on most modern operating systems. It is currently being developed and tested on a virtual machine running Debian Linux 6.0 (Squeeze) on VMWare ESXi 4.1.

Users of this application are not limited to Linux or any major web browser. Any computer capable of running a web browser that supports basic HTML and SSL, which includes all of the most popular mobile and PC browsers, will be able to access the application.

2.2 USER STORIES

These are the general requirements behind the project that were given to us by the product owner. These are non-technical and should be understood by all parties.

User Stories			
ID	Description	Priority	Cost
1	As a user, I would like to be able to visit and access all pages of my website.	10	20
2	As a user, I would like this program to record any resources that are unavailable, including dead links, missing images, scripts or CSS files.	10	20
3	As a user, I must be able to view a report of a given scan. This report should show all broken links and missing images that fall under my domain. These scans should be listed and accessible from a website.	10	30
4	As a user, I would like to view scans and start new scans from a website.	10	20
5	As a user, I would like to be able to sort the reports that are presented to me. Useful sorting functions would be: by the order in which the pages were visited, alphabetically, and alphabetically by the parent page.	7	10
6	As a user viewing a report, I should be able to generate that report in a printer friendly format.	6	10
7	As a user, I would like to be able to select two scans and show only the items that have changed.	6	6
8	As a user viewing a report, I should be able to view reports from scans that are in progress	6	4
9	As a user, I would like to be able to limit the run time of a scan when I start it, either by time or by distance from the start page.	6	12
10	As a user, I would like to select a scan, and run a new scan that will check if the previous errors have been corrected.	5	10
11	As a user, I would like for reports to include pages that are accessible over secured links.	5	8
12	As a user, I would like to sort a report based on the subdomain.	3	14
13	As a user, I should have to login before initiating a scan or viewing a report.	3	2
14	As a user, I might like to pause a scan that was currently in progress.	2	4
15	As a user, I would like to have scans that were automatically run at regular intervals.	1	6
16	As a user, I would like to see page load times in my reports.	1	2

2.3 USE CASES

These technical use cases come from the above user stories and are used to determine what gets done in a project cycle.

ID: Crawler-1

Priority: 10

Cost: 20

Name: Basic Web Crawler

Description: The system should be able to visit all of the pages on the website.

Precondition: Report-1 (A scan has been initiated.)

Primary flow:

1. The web crawler will retrieve the root page of the domain.

2. The web crawler will create a page object with the relevant information and place it in a pending queue.

3. If there is a page object in the pending queue, it will be popped, and become the current page. If there is not, the scan is finished, and the crawler exits.

4. The web crawler will parse the retrieved page for resource URLs and insert new page objects into the pending queue if they have not yet been visited and are not already queued.

5. Return to 4.

Alternate flow:

1. The web crawler will initialize a visited list and a pending queue.

2. The web crawler will retrieve the root page of the domain.

3. If the root page cannot be retrieved, the scan exits.

Postconditions: Every URL referenced in any page reachable from the root page will be visited. This does NOT guarantee that it will be retrieved, as it may not exist.

ID: Crawler-2

Priority : 10

Cost: 20

Name: Record Crawler Results

Description: The results of crawling the website will be recorded in a database so that they can be used later.

Preconditions: The webcrawler must be running. The webcrawler must have generated a record in the "scans" table and retrieved a scan ID.

Primary flow:

1. When a page object is created, a matching entry in the database will be created, including the scan id, a link to the parent page (this can be null for the root of the scan), the current URL, the retrieval time, and the HTTP response code.

2. The resource_children table will be updated to create a link between the parent and the new child object.

3. The resource id of the newly created record will then be stored in the page object.

Alternate flow:

1. When a child resource is found to already have an associated page object in the pending or visited queue, a link will be created in the resource_children table between the current page object and the one that was found.

Postconditions: The resources table will contain a complete list of all reachable nodes on a graph of the website. The resource_children table will contain a complete list of the reachable edges of a graph of the website. The resources table will also contain results codes, retrieval times and a spanning tree of the graph of the website. As the response codes will be recorded and records will be created even for pages that are not accessible, this will be a record of broken links as well as working ones.

ID: Report-1

Priority : 10

Cost: 20

Name: Crawler Interaction

Description: The user must be able to start new scans and list the completed scans.

Preconditions: ID-13 (Login)

Primary flow:

1. The system will display a button to start a new scan.

2. The system will retrieve a list of scans that have been completed.

3. The system will display the list of completed scans.

4. The user may click on the "Start Scan" button.

5. When the button is clicked, a scan will be dispatched via a call to exec();

Alternate flow:

1. The system will display a button to start a new scan.

2. If there are no scans completed, none will be listed.

3. The user may click on the "Start Scan" button.

4. When the button is clicked, a scan will be dispatched via a call to exec();

Postconditions: The user will know what scans have been completed. If the button was depressed, a scan will have been started.

ID: Report-2

Priority : 10

Cost: 30

Name: Show Scan Results

Description: Reports of the scans will be visible to the user, showing all irretrievable resources under the specified domain.

Preconditions: Crawler-2 (A scan must have been recorded), **Report-1** (the user must have a list of scans.)

Primary flow:

1. The user will select a scan that has been run.

2. The scan container will open.

3. The scan container will request the scan data from the webserver.

4. The webserver will retrieve all page resources that match this scan id, format them as JSON, and return them to the scan container.

5. The scan container will then display the JSON results.

Alternate flow: None

Postconditions: The scan will be displayed.

ID: Report-3

Description: The page will offer buttons to sort the reports by the order in which the pages were visited, alphabetically, and alphabetically by the parent page.

Precondition: Crawler-2 (A scan must have been recorded.)**, Report-2** (A report must be displayed)

Primary Flow:

1) The system will retrieve data from the database.

2) The system will sort data based on the sorting order selected by the user.

3) The system will display the results of the sort in the website's UI.

Postcondition: The system will display reports sorted in the manner selected by the user.

ID: Report-4

Description: Reports will be displayed in a printer-friendly format.

Precondition: Crawler-1,Crawler-2,Report-1, Report-2, Report-3

Primary Flow:

1) The system will retrieve data from the database.

2) The system will sort retrieved data.

3) The system will display sorted or unsorted data in the website UI.

4) The system will parse the data generated in the web UI.

5) The system will create a printable text based format of the data displayed in the website's UI.

6) The system will display the printable version of the scan results in a new window.

Postcondition: The system will generate a new window displaying a printable text based format of the results of the scan.

ID: Report-5

Descripton: The system will be able to select two scans and show only the items that have changed.

Precondition: Crawler-2 (There must be a scan.), **Report -2** (The user must have a scan open.)

Primary Flow:

1) The system will be able to retrieve the results of two scans.

2) The system will be able to compare the results of the two scans.

3) The system will be able to display the results of the two selected scans that do not match.

Postcondition: The system will have shown the differences between two scans.

ID: Report -6

Description: The system will be able to display reports from scans that are in progress.

Preconditions: Crawler-1 (A scan is currently in progress.)

Primary Flow:

1) A scan will be started by the system.

2) The system will be able to retrieve data from the database while a scan is in progress.

3) The system will be able to display data from the database in the web UI while a scan is in progress.

Postcondition: The system will be able to retrieve data from the database while a scan is in progress.

ID: Crawler-3

Priority: 6

Cost: 12

Name: Runtime Limit

Description: The scan can take a long time to fully run. As a result, this use case is designed to allow a user to limit the run span of the scan. The user can decide to have the scan run as much as it can in a certain amount of time, or to limit the scan to only run until it gets to a certain distance away from the main page.

Pre-Condition: **Login** (User has logged in.)

Standard Flow:

1. The user decides how he/she wants to limit the search (either length or depth) and enters the appropriate value in the appropriate location for the choice. This starts the scan.

2. The scan will run as normal and go through all of the pages until it reaches the limit set by the user.

3. The finished report with all the errors for the pages searched will be displayed.

Alternate Flow 1:

1. The first step will be the same as the standard flow.

2. The scan will run until the user pauses the search. This will also pause the timer if the

user chooses to do a time based search. The scan will remained pause until the user resumes the search.

3. Same as step 4 in the standard flow.

Alternate Flow 2:

1. The first step will be the same as the standard flow.

2. The scan will run until the user stops the search. This will terminate the search.

3. The final report will be printed and displayed all of the errors up to the termination of the search, which will not be the same as what the user initially requested.

Alternate Flow 3:

1. The user enters invalid information for one of the two limiting factors trying to be used.

2. An error message will be displayed informing the user that the information entered is incorrect and prompt them to try again.

3. Return to step number 2 in the standard flow.

Post Conditions: A scan will be completed based on the limiting factor given by the user. A report will be completed and displayed for the user to examine. A record of the scan will be saved in the database so that it can be accessed in the future, used to be compared to other searches, or used to start another search to check for errors being fixed.

ID: Crawler-4

Priority: 5

Cost: 10

Name: Error Check

Description: This use case will select a scan that has been run previously. A second scan will begin and will only rescan what was scanned in the previous results that were initially selected. It will return a report and show a comparison of the two reports to show what errors are the same, what errors are new, and what errors are fixed (if anything has been).

Pre-Conditions: Report-1 (The user is looking at the list of scans.)

Standard Flow:

1. The user will select a previous report that he/she wants to see if any changes were made.

2. The user will start a comparative scan.

3.The scan will run and scan the same pages as the previous scan.

4. A report will be generated based on the scan that was just run.

5. The two scans will be compared, showing what errors were still present, any new errors that popped up, and showing which errors were fixed.

Alternate Flow 1:

1.The first 3 steps from the standard flow will be followed.

2. The user will pause the scan during the search.

3. A partial report will be shown, displaying what errors have been encountered thus far.

4. The scan remains paused until the user resumes the scan. The scan continues on normally.

5. Return to step 5 in the standard flow.

Alternate Flow 2:

1.The first 3 steps from the standard flow will be followed.

2. The user will stop the scan during the search. This terminated search.

3. A report will be created based on the partial scan, and the report will compare only the parts of the scan that were completed.

Post Condition: A repeat scan will have run. There will be two reports based off of the same scan. They will be compared and the user will see what. If anything, has changed for either better or worse.

ID: SSL

Priority: 5

Cost: 8

Name: Secure Check

Description: As the web crawler goes through the SPSU domain, it will come across links that are secure. These items should be listed as secure login links and not as errors (as they can not be accessed) and shown as such on the report that is displayed at the end.

Pre-Conditions: Crawler-1(A scan is running.)

Standard Flow:

1. User starts a scan.

2. Web crawler and error reporter are called and begin to work.

3. As the crawler tries to access something in the SPSU domain that requires a secure login, it will report that to the database as a link to a secure site.

4. When the scan has completed, the report will populate all of the items from the scan.

The secure links will appear on the report, and will be listed as secure links instead of errors.

Post Conditions: The user will have an updated report that no longer lists the secure pages as a link that can't be accessed, but as a secure page that requires a login.

ID: Report-7

Priority: 3

Cost: 14

Name: Subdomain Sort

Description: When looking at the report, we will be able to sort it based on the subdomain. This will allow us to break down the report and look at the errors only in certain subdomains, which would be useful to do something along the lines of finding all of the errors in a certain department's webpages.

Pre-Conditions: Crawler-2 (A scan must have been recorded.)**, Report-2** (A report must be displayed)

Standard Flow:

1. User starts a scan and receives a report, or opens a report that has been previous run and recorded.

2. User clicks on the sort feature that says subdomain.

3. A sorting algorithm (using pattern matching) is used to arrange the items in the report based off of their subdomain. The default case will be alphabetical. If the button is pressed twice, the report will be in reverse alphabetical order.

Post Conditions: The user has a viewable report that properly lists the items based on their subdomain either from a-z or from z-a.

ID:Login

Name: User Login

Priority: 3

Cost: 2

Description: This is a security feature that prevents unauthorized users form making the product into a Denial of Service attack platform or find information about the target website that might be exploitable. This login will be set on creation of the system. The login will require a user name and simple password. The login information will be stored in the data base.

Preconditions: None

Standard flow: User will be prompted for user name and password upon arrival to the website. The user will then type in a user name and password into there appropriate blanks and press the login button or enter.

Alternative flow: IF the user does not enter a field such as password or user name the login will fail. If user enters a wrong user name or password the login will throw a invalid login error. Should the user try and bypass this step web pages beyond this will not be displayed.

Post Conditions: After a successful login the user will be able to go beyond the login page and view or start scans.

Considerations or issues: This is a low priory by the customer however for security reasons needs to be implemented. There also is no registration now and requires the system admin to add new login entries.

ID:Crawler-5

Name: Pause Scan

Priority: 2

Cost: 4

Description: This will temporally pause the web crawler aspect of the scan. This will not stop processes such as report builders. This option is only available after a scan has started. Once the scan has been stopped the user can continue scan at the press of a button.

Precondition: Crawler-1 (A scan is running.)

Standard Flow: User has started a scan and then presses the pause scan. The scan stops and can be restart by pressing continue scan.

Alternative Flow: Scan is unable to restart so error is thrown. User is able to access stop scan before scan has started do nothings.

Post Conditions: Scan continues.

Considerations/ Issues: This might include pausing threads.

ID: Crawler-6

Name: Timer

Priority: 1

Cost: 6

Description: This allows a user to run a scan at a user set time. This involves the server knowing what time it is and starting a process based on that. The scan will not display a live report unless a user has logged in during its run. This can be set up to run once, weekly or monthly.

Precondition: Login (User has logged in)

Standard Flow: A user has logged on and imputed a time and date in the required box and check the required boxes dealing with now often this reoccurs. Should no options be checked and no date and only a time the default is run once.

Alternative Flow: Should the user input nothing no scans will be scheduled. Should an impossible time be imputed no scans will run and a message will display to the user that this is an invalid time.

Post Conditions: A scan will run

Considerations/ Issues: This can be used to make a Denial of service attack if abused. However this scan could be slowed down should this become an issue.

ID: Crawler-7

Name: Stopwatch

Priority: 1

Cost: 2

Description: This will add into the report the time it took to load a page.

Precondition: Crawler-1 (A scan is running.)

Standard Flow: During the web crawler's execution times will be recoded of the load times of web pages.

Alternative Flow: None

Post Conditions: There is now a field in the data base that holds web load times

Considerations/ Issues: Adds a new field to the data base which could potential slow down the scan.

3. DESIGN

3.1 SYSTEM ARCHITECTURE

The Forager system consists of two parts. The webcrawler is written in Python3, using the requests library (a wrapper around the native urllib3 HTTP client). It uses PsycoPg2, a PostgreSQL connector that supports the DB-API interface defined in PEP 249. The report viewer and user interface is written in PHP 5, which is served by an Apache 2.2 Web server. Scan results are stored in a PostgreSQL 8.4 database, and all components are deployed on a machine running Linux.

A user wishing to interact with the system will go to the login page and enter their authentication credentials. They will then be presented with a menu allowing them to either interact with the crawler or existing scans. Crawler interaction is mediated with the control_json.php page, which will take several options, allowing it to start a new scan, as well as check the status, pause, or stop a currently running scan. When executed, this page will check the database for the PID of any scans without a stop time and attempt to send a liveness query (SIGUSR2) to them. If any of these signals succeed, this PID is assumed to be that of the currently running scan. Any PIDs where the signals fail are marked in the database as having exited. The script will then either send the signal requested to the running process (SIGUSR1 is used for pause, and SIGCONT for resume), or exec a new process. The PID of the currently running process is then returned in a JSON document, or -1 if there is not one. On startup, the crawler will initialize its database connections and logs, and then use a simplification of Richard Stevens' daemon initialization algorithm to cleanly detach from the console. It will then create a scan record in the database, storing its process ID and the time that it was started.

The crawler uses a resource object to represent the URLs that it is given. When the crawler is initialized, it creates an object for the initial URL and stores a reference to it in a hashtable of existing resources and in a pending queue for objects that have not yet been retrieved. If this is a new scan, this object initially contains only the URL and a null link to its parent. If this is a rescan of an existing report, the queue is initialized with the failed links from the base scan. The crawler then processes the pending queue until it is empty by retrieving the first element, using the resource's fetch() method to visit the page and store relevant information, like the HTTP response code and the load time. If the resource is an HTML page, and the scan is a new one, the HTML is parsed and a list of children is stored in the resource. The resource's SQL_Call() method is then used to store the resource in the database, whose row structure matches the object definition. If any children were found in the resource, the crawler will iterate over them, and any that do not already exist in the resource list have resources created for them and are placed in the pending queue and the resource list. When the pending queue is exhausted, all resources that meet the restrictions of the crawler and that are reachable from the first page will be stored in the database, and the parent records will describe a spanning tree of the site map. When the queue is exhausted, the crawler will store its exit time in the database and shut down.

Once a scan has been registered in the database, it will be visible from the scans page on the website. The data is retrieved from the database and converted into JSON

(JavaScript Object Notation). This JSON data is loaded by jQuery and processed into a sortable table with the DataTables jQuery plugin. Each of the scans can then be clicked to retrieve a list of the URLs visited in a similar manner. The list will show and allow sorting based on the response time, response code and URL. When the user opens the controller window, they will be able to stop, start, pause or resume scans. While a scan is running, the contents of that scan will be continually queried by a javascript function and added to the end of the table. The user will also be given the opportunity to compare scans, which will query the database for the contents of both scans and return any URLs that received different return codes.

4. MANAGEMENT PLAN

4.1 PLANNED ASSIGNMENTS AND SCHEDULE FOR FIRST CYCLE

Group 4's planned assignments for cycle 1 will breakdown as follows:

Week 1:

The Team Assignments
Gather requirements
Tune User Stories
Assignment of tasks

Individual Assignments
Matthew Powell - Requirements
Robin Mays - Requirements
Thomas Couture - Requirements
Samuel Hall – Requirements, Set-up server

Week 2:

The Team Assignments
Revise requirements
Tune User Stories
Weekly Status Report

Individual Assignments
Matthew Powell – Coding, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

Week 3:

The Team Assignments
Weekly Status Report
Project Analysis Report
Project Demo

Individual Assignments

Matthew Powell – Coding, Testing, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

4.2 ACTUAL ASSIGNMENTS AND SCHEDULE FOR FIRST CYCLE

Week 1:

The Team Assignments
Gather requirements
Tune User Stories
Assignment of tasks

Individual Assignments
Matthew Powell - Requirements
Robin Mays - Requirements
Thomas Couture - Requirements
Samuel Hall – Requirements, Set-up server

Week 2:

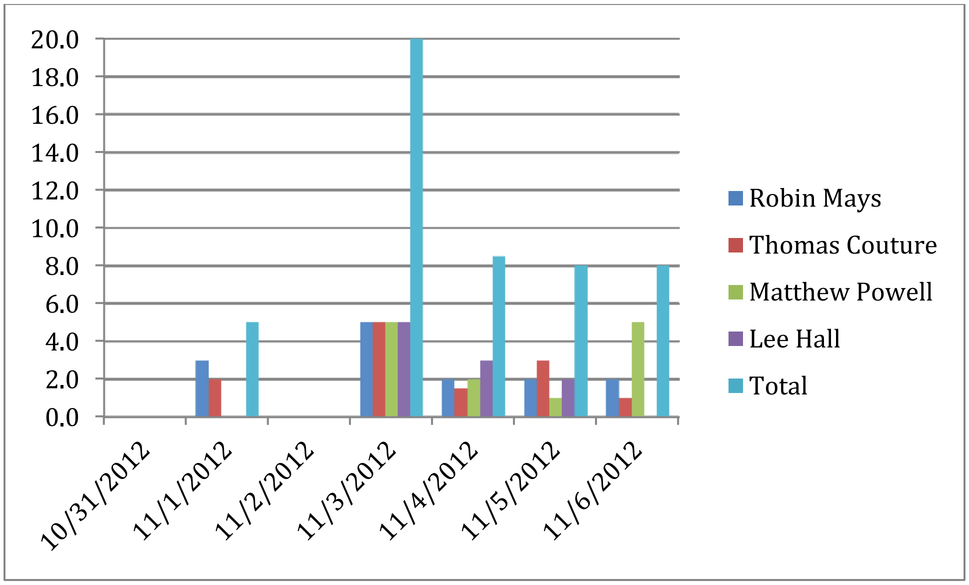
The Team Assignments
Revise requirements
Tune User Stories
Weekly Status Report

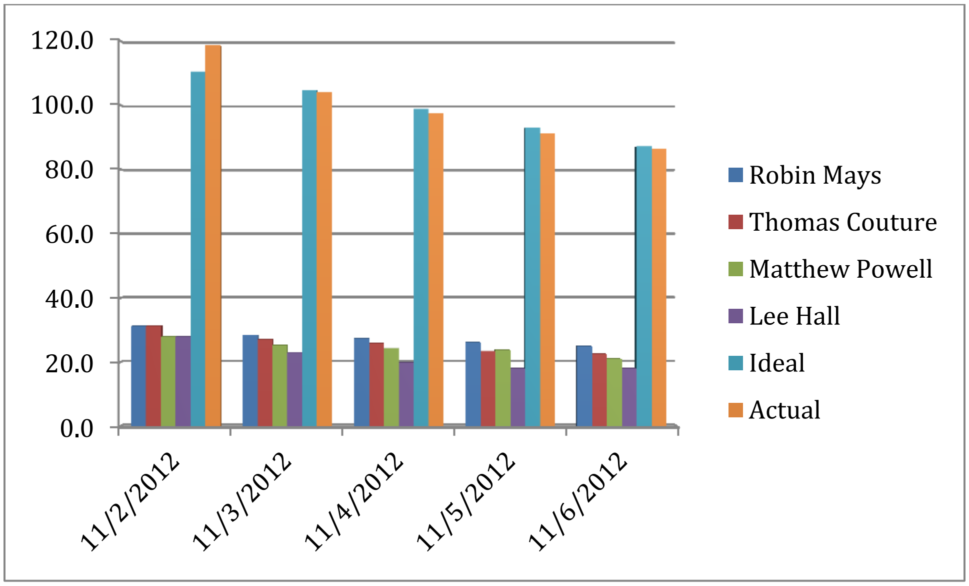
Individual Assignments
Matthew Powell – Coding, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

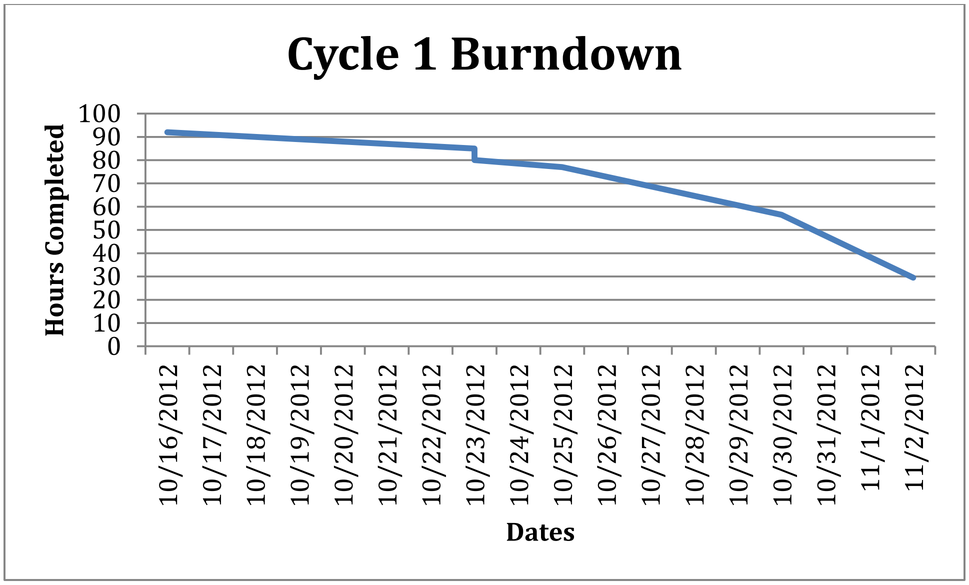
Week 3:

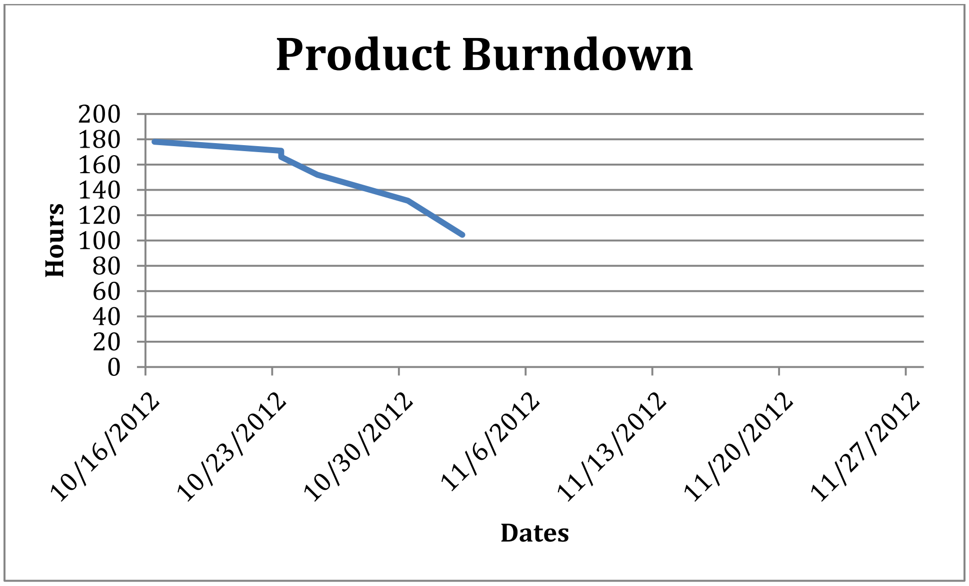
The Team Assignments
Weekly Status Report
Project Analysis Report
Project Demo

Individual Assignments
Matthew Powell – Coding,Testing, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation


20.0
18.0
16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
10/31/2012
11/1/2012
11/2/2012
11/3/2012
11/4/2012
11/5/2012
11/6/2012
Robin Mays
Thomas Couture
Matthew Powell
Lee Hall
Total


120.0
100.0
80.0
60.0
40.0
20.0
0.0
11/2/2012
11/3/2012
11/4/2012
11/5/2012
11/6/2012
Robin Mays
Thomas Couture
Matthew Powell
Lee Hall
Ideal
Actual


Cycle 1 Burndown
Hours Completed
100
90
80
70
60
50
40
30
20
10
0
10/16/2012
10/17/2012
10/18/2012
10/19/2012
10/20/2012
10/21/2012
10/22/2012
10/23/2012
10/24/2012
10/25/2012
10/26/2012
10/27/2012
10/28/2012
10/29/2012
10/30/2012
10/31/2012
11/1/2012
11/2/2012
Dates


Product Burndown
Hours
200
180
160
140
120
100
80
60
40
20
0
10/16/2012
10/23/2012
10/30/2012
11/6/2012
11/13/2012
11/20/2012
11/27/2012
Dates

5. RISK MITIGATION

5.1 RISK MITIGATION

This data in the below table and chart will be used to help reduce the damage and danger of risks that could potential delay or give unwanted functionality to this project.

ID	Risk	Mitigation
1	We might have issues with SSL support in our HTTP Request library. As previous projects have had issues with this, the likelihood of some impact is fairly high, but the impact is relatively low as the user story is a low priority.	This can be mitigated by research. Appropriate design modularity should also allow us to swap out the Request library without much trouble if we run into problems.
2	We could have communications problems with the report viewer and the web crawler. As they are written in different languages and running as completely distinct processes, we could potentially run into problems moving data between them. This is a relatively low likelihood, as database communication in python and php are both well-trodden ground, but the impact would be severe.	A backup plan might be to use python to generate JSON directly from pickled object from the web-crawler, or potentially generate one shot reports directly in the web-crawler, though these options are both less flexible.
3	We could have integration problems between the web crawler and the web interface. Other projects have had issues communicating between the two pieces, and being able to start the webcrawler from the web interface is a high priority.	Research is again a key defense against this failure, as these problems have certainly been widely encountered. Richard W. Stevens's Advanced Programming in a Unix Environment, while written in C, covers many of the pitfalls of IPC and daemonized processes in this environment, and careful reading should produce solutions to the most common problems which can be adapted for the languages in question.
4	We could have concurrency problems with the asynchronous communication between the web crawler and the web interface. This seems like a very low risk proposition, as updates only come from one process and receiving data that is a few seconds stale does not have any user impact.	This can be ignored.

	Impact								
Risk	1	2	3	4	5	6	7	8	9
9	🟨	🟨	🟨	🟨	🟨	🟨	🟥	🟥	🟥
8	🟨	🟨	🟨 ID1	🟨	🟨	🟨	🟨	🟥	🟥
7	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨 ID3	🟥
6	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
5	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
4	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
3	🟩	🟨 ID3	🟨	🟨	🟨	🟨	🟨	🟨 ID2	🟨
2	🟩 ID4	🟩	🟨	🟨	🟨	🟨	🟨	🟨	🟨
1	🟩	🟩	🟩	🟨	🟨	🟨	🟨	🟨	🟨

6. CYCLE POST-MORTEM ANALYSIS

6.1 SUCCESSES

Group 4 welcomed many successes during the course of Cycle 1. They included:

The creation of usable user stories and their conversion to use cases that better defined the project.

6.2 FAILURES

Time management was significantly less than optimal due to communications failures during the creation of use cases. The use of email in this process was inefficient and cumbersome, causing communication to be limited and unproductive. In the future, GitHub will be used to better share documents through the internet and meetings will be held in-person, as that has proven to be the most productive use of the team's time.

6.3 LESSONS LEARNED/RISK MITIGATION

The failure to construct use cases in a timely fashion led to a late start of the project coding. However, when the use cases were rewritten, in most cases it was because they were too in-depth, providing algorithmic details that helped the actual implementation of the use cases. This is a situational problem and cannot be counted on in the future to work out as a net positive, so the plan outlined in 6.2 should be implemented so that this scenario does not happen again. After this process had finished, we also had a better understanding of the fundamentals of use cases, which will allow us to avoid this problem in the future.

Another risk that was seen this cycle was a lack of comments in some sections of the code. There were no negative consequences resulting from this in the current cycle as paired programing and good communication mitigated the risk. However, this might not be the case so in future cycles, so more comments will be strongly advised.

7. TEST PLAN AND PROCEDURES

7.1 TEST PLAN

7.1.1 Introduction

The test plan for the Forager project will include verifying results of the web crawler and the correctness of the information in the reports. This will be done in such a way to detect errors in the web crawler's data collection as well as potential avenues for abuse of the web crawler or its user interface. This will also cover checking for misleading or inaccurate information and the usability of the web interface.

7.1.2 Test Items

The test will cover all use cases started and or completed in this sprint as well as potential avenues for abuse and non-intended functionality. However, the comprehensive testing will be done on a fully functional project and not in parts. This will test both the integrity of the parts as well as the integration. Due to the nature of this project testing involved in finding ways to increase speed of the scan will not be undertaken because of potential risk to the SPSU webservers.

7.1.3 Software Risk Issues

As stated above, there is risk that this program could inadvertently cause a denial of service attack on the SPSU domain. This risk has been mitigated by a policy of not running the web crawler during normal operating hours. We also must take into account that the server running this service has other functions and hosts services of its own, which limits most forms of aggressive penetration testing.

7.1.4 Approach

Testing of User Interface (UI) elements will be done with the 3 most common web browsers: Google's Chrome, Microsoft's Internet Explorer, and the Mozilla Foundation's Firefox. Malicious non-UI input will be tested with the TamperData extension to Firefox and direct access via telnet. Items will be considered to pass if the application behaves as expected. In response to legitimate user input, the application should either take the action requested, provide the user clear instructions on how to proceed, or notify the user and the system administrator of an uncorrectable failure of the application. In response to abusive input that cannot be accomplished directly from the user interface, the application should refuse to leak information. Useful information may be provided when it does not leak information, but generic failure messages are also acceptable. The test results will come in order that a normal user would interact with the system.

7.2 TEST RESULTS

7.2.1 Login

Login credentials can be guessed, however page data does not release any data that it should not. Once logged in, the user cannot log out until browser session is closed.

7.2.2 Main Page

The user is able to go to the “start a scan” and “view reports” page from here. However “compare reports” and “extra” give 404 errors. These sections have not been implemented, so the error is to be expected at this time. It would also appear that there are links at the bottom of the page that do not do anything. These will be most likely removed, and at this time pose no risk. After viewing this page information, no data was found giving the user information that they should not have.

7.2.3 Scan

When arriving at the scan page, the scan starts automatically. This however cannot be stopped via the web UI and needs to be changed. It is noted that 2 scans cannot be run at the same time.

7.2.4 Reports Main Page

When arriving to this page, the first 10 scans are displayed. The show # of entries bar works as intended and cannot be changed with the program tamper data. Tamper data the add on for firefox is unable to interact with any of the fields on this page. The search bar filters results as opposed to refreshing the page and works as intended. Clicking on a scan ID # will direct the browser to a new page. The same links at the bottom of the page as seen in the main page will send the user to the top of the page without reloading it. All other links from this page work as intended, and as previously seen on the main page. It should be noted that I can sort by ID, Start Time, End Time and Run Time. These sorts appear to work as intended. The end time and run time for the scans do not show up. The exception is some user created data that is not from a real scan. The start time has unnecessary noise in it as well.

7.2.5 Report Page

“Show # entries” works as intended however there is not next page option available and the maximum number of entries per page is 100. A full scan has 3,467 entries. The search bar works as intended and can take both #s to search IDs and strings to search the URLs. In some cases, the sorting causes long URLs to display outside of the intended area. This effect can be found when the user sorts by HTTP Response, having code 999 on the top and setting entries to 100. It should be noted that the user cannot go from this page directly back to the view reports using the links bar in the banner. I am still able to go to the main page or use the back page function to navigate. The links at the bottom return the user to the top of the page.

7.2.6 Crawler verification

After viewing a report some of the URLs were checked in the browser. All negative HTTP responses were reachable but required login credentials. Responses with 200 were reachable and all 404 and 999 were unreachable. This was using a random sampling form the results found.