Contents

Management Plan To keep track of how much progress has been made toward meeting user requirements, and to document how much each member of the team has contributed, the PHP team plans to use TRAC to develop tickets for hour reporting, PHP doc to demonstrate number of lines of code written, and GitHub to demonstrate trends in the number of lines of code each person wrote.

0.1 Tickets through TRAC

http://www.nwerp.org/trac/gus/report/6
TRAC is a valuable tool that not only communicates when a task should be completed, but also who was assigned to complete the task, and who actually completed the task. The testing/documentation subteam has and will continue to generate tickets through TRAC by:

- identifying the tasks to be completed
- estimating the number of hours required to complete the tasks
- recording the responsible individual to complete the task

The responsible individual then will:

- work on their tickets, to include writing unit-tests and writing a PHP docstring to the unit testcase specification template
- submit the code to the repository
- record the number of hours spent writing the code

1 Test Plan

http://www.nwerp.org/gus/index.php/test
Each software developer will be responsible for writing use cases to demonstrate the functionality of their code. PHP Coverage (Black Duck Software) will be used to demonstrate all the lines of code have been tested, and PHP Unit will be used to write the unit tests. As the semester progresses, the testing/documentation team will cross-reference the unit testcase specification templates into the test management plan.

1.1 Unit Testing Template

http://wiki.services.openoffice.org/wiki/Test$_{c}ase_{s}pecification_{t}emplate$
$Eachsoftwaredeveloperwillusethetestcasespecificationtemplateidentifiedinclasstowritetheirtestcases.$

have the developer's name on top

cross-reference the use-case in the SSRS

cross-reference the page illustrating the use-case

cross-reference the page showing the results of the unit test

document what was tested

1.2 Code Igniter Unit Testing

http://codeigniter.com/user$_{g}uide/libraries/unit_{t}esting.html$
$Toautomateintegrationandstructuraltesting, eachunittestwilluseCodeIgniterUnitTesting(wherepossib$

1.3 Integration and Structural Testing

The testing/documentation subteam will complete a structural regression test every sprint to ensure that all of the GUS components work together. This will include an assessment of Red/Green Light testing, and tracking which tests have passed over time. The PHP Doc/Test team will review each set of tests each scrum to determine that they have sufficient depth. As functionality of GUS grows, superior classes will aggragate basic classes, and the unit testing of the superior classes will be sufficient enought to qualify as Integration Testing.

1.4 System Testing

To ensure the customer's requirements are met, PHP will complete functional, performance, acceptance, and installation testing. System Testing will be conducting through development as the GUS system is being used for actual project planning. Bug will be recorded, delegated, and fixed using the Trac system linked to the GUS development.

1.5 Functional Testing

To the extent possible, PHP will include scripts to simulate GUS users completing the use-cases stated in the SSRS. Each function in the SSRS Use Cases will link to the test and the actual location in the code where it is implemented.

1.6 Performance Testing

Using the scripts generated for functional testing, PHP will test the **scalability** of GUS to the estimated load of 10,000 users (1000 actively involved) 500 of which will be actively logged in at any given time. The number of groups is easier to estimate at 200 groups, 20 of which are actively involved with ASUI. The Security Subteam will write scripts and controllers to test the system, and have respective tests to pass or fail. This will include web robots and perforance logging during development.

1.7 Usability Testing

PHP continuously completes alpha tests through the use of the testing/documentation team. The testing/documentation team completes an alpha-type usability test every time they take a screen-shot to make an updated entry in the user's manual. Each significant release of GUS will be compared with the Customer Requirements in the SSRS, either directly mapped or loosely. Trac Milestones containing all Major Customer requirements are set, and those requirements are further drilled down into billable tasks with Trac tickets. PHP will complete at least one beta test with university groups. Ideally, we could beta test with ASUI. One of the groups that has already agreed to complete a beta test is Lutheran Campus Ministry.

2 Metric Plan

Metrics will include:

- PHP Coverage (Black Duck Software)
- Red/Green Light Tracking
- Trac Billable Hours
- Github Impact Statements
- LOC tracking
- SSRS/Trac Customer requirements met

  Coverage testing will be run once a week, as well as Red/Green Light Tracking, as soon as an automated system is built to run them(Scrum 8?). Trac Billable Hours will be used weekly (starting Scrum 6), as well as LOC Tracking. Github Impact Statements

have been active since Fall 2010. SSRS/Trac Customer Requirements are also in use, and will be full defined (Scrum 8?).