1 Testing Plan

1.1 X3Profiler's Policy On Testing

Whether we like it or not, testing is an important part of the software development process. It will let us sleep comfortably at night knowing that our software does what it should be doing. Now, RUP has several core practices for handling Testing as methodically as possible and from this set of core practices we have selected those that will actually come in handy for our project.

Namely, we will document our bugs (at least the most outstanding ones, which are presented later on in this document) along with their respective fixes. We shall also utilize the requirements document as a check to verify that the end result of our program is what the user expects us to provide.

We will focus our testing primarily on the three main operations done by X3Profiler, which are parsing, graphing and displaying. Just as RUP recommends, during our testing phase we actually want to break the damn software. We want to make our software cry many times over before we are through with it. Thus, we have targeted the following areas as potentially problematic:

1. Parsing gprof input. Whenever we extract data from something and condense it, there is a potential for mistakes.

2. Graph displayed does not faithfully represent data in gprof file. Maybe we sent in the wrong data into PYX though we correctly parsed it.

3. Graph displayed does faithfully represent the data found in the gprof file, though it was not in the format or operation requested by the user.

1.2 Automatic Tests

We have included in our submitted files (also available in our CVS repository and website) a collection of short, automatic tests that can be used to verify X3Profiler is working as it should be working. Most of the tests themselves check for internal data consistency, broken lists, incomplete data sources, etc. We have decided, however, to not only have these tests as complete "standalones". We have integrated some of them into our code since they add to our stability and reliability. These tests are normally commented so if you wish to check them out, grab the files and read on.

Most of our automatic tests are in the form of Unit Tests, since we can not really perform full fledged automatic System testing in our graphical environment[1]. We also have several, though not many, Integration Tests in which we verify whether out DataTypes (the heart and soul of X3Profiler) are working well with other pieces of the program.

1.3 Manual Tests

As you might have noticed, our project's main goal is to take a nasty textual representation of a gprof output file, and making it all sexy and visual. Now, given the nature of our project, there

[1] we have made up for this with an extensive manual test section

really isn't much automation our tests. We have several automated test to check for internal errors, though these do not form the bulk of our testing efforts. What we have here is a set of gprofiles (we have only included the flat profile since it is what we actually use) and a set of graphics.

Our testing methodology for this part is to take the gprof file, graph it inside the X3Profiler program, and then compare it with the file we have supplied here. We have taken the effort to manually revise each of these graphs with the gprof output to ensure that they faithfully represent the data. Some of the graphs are harder to make out than others, but thats the nature of the game given the common low run times generated by gprof. Now, using these supplied sets of graphs you can authenticate that your freshly downloaded version of X3Profiler actualy generates the correct results. In a way, we are doing a semi smoke test (in the sense that it ensures that any possible changes done to the program will not break any other parts.)

Do note that while we do not test *ALL* the graphing functionalities and combinations of profiles, we actually do test both main sources of errors. Let me explain here before you say "what the hell did he just say?" So, our program is built upon the graphing library *PYX*, as such, our testing goals finish as soon as we can verify that we are "feeding" PYX the right information taken from the gprofiles. While this might sound like an incomplete testing job, well, it really isn't our job to test PYX itself. We do ensure that we do not have any bad input for PYX, but by itself, we can not alter PYX.

Thus, our manual tests do the following:

1. Check that each individual gprof file is parsed correctly (this is done when you add a new "atom" to the program. It will display the parsed information on the right pane, so a quick check between that pane and the actual gprof file will tell you clean off whether the parser module is doing its job or not.)

2. Check that each individual atom can be correctly graphed based on Number of Calls, Percent Time, Cumulative Seconds, Self Seconds. By ensuring that each individual atom can be correctly graphed based on one operation from the gprof file we ensure that the correct information from the parsed profile is used.

3. Check that each different graph comparison format is correctly graphed. This includes the Error Chart, the Bar Graph Compare, the Stacked Bars, and the Differential Graphs.

As you can see, by doing it this way we are ensuring that we always feed into PYX the correct information regarding whether we want say Number of Calls, Percent Time, etc. or whether we want a Differential Graph or a Stacked Bar Comparison. Thus, by independently testing each possible stream of options the user has, we actually do cover all the ground in the middle that we do not *explicitly* test here.

1.4 Log of Most Notable Bugs

1.4.1 The Bugs

There were some interesting bugs (if they can be called that) that surged from the manual testing. Other than the occasional typo and wrong variable usage (which are common mistakes

not linked to logical errors), we had some issues with PYX itself, QT and just pure old Python. Namely we got errors based on the following:

1. If the value being graphed was either really small (close to 0), or equal to 1, PYX would crash throwing out an error about "math error range in logarithm scales". (Found by Greg Lazarus.)

2. If we used file writing to input data into PYX (which was simpler to adapt from the already existing PYX code examples) we ended up with graphs that would not update. This was not truly a bug and more of a hidden feature that PYX hides under the rug. (Found by Andres Ramirez)

3. If the values being graphed are all equal to each other, PYX will conglomerate them into just one big chunk. Not pretty. (Found by Andres Ramirez)

4. QtStrings are just pure evil. They are related to Michael Jackson. They had conflicts with Python's Pickle and Marshalling systems. Hell, it even *segfaulted!!!! Python.* Enough Said. (Found by Robert Rypka)

5. First of all, Python's Pickle had conflicts with QtStrings. Later on Qt released an update to their source which ended up breaking our program for one of our developers[2](Found by Greg Lazarus.)

6. If the gprof file we supplied reached roughly 1GB in size, it will crash the X3Profiler parser. (Found by Robert Rypka)

1.4.2 The Orkin Man (aka, the fix)

For bug # 1, we simply had to maneuver around the source of the problem, PYX and its axis scaling. See, PYX tries to automatically generate scales for the data it is supplied. While this seems like an amazing idea for us (less work!), it proved to be a nasty error to find. Think about it for a sec, Python will spit out about 20 lines of cryptic errors on logarithms from PYX... but wait, we don't explicitly do anything with PYX.... WTF. It then realized to our mathematical geniuses that the logarithm of say, 1, is 0. Great, how can we have a scale of length 0? So, we had to completely circumvent PYX's automatic range generation if we wanted to avoid any logarithm problems. So, we decided to find out what the maximum value of the profiled data was (while we were parsing it into X3Profiler) and then made a *linear* range from 0 to $\frac{3*max}{2}$. This resolved this graphing problem at last.

We spent close to 3 weeks trying to figure out what the hell caused bug # 2. We were baffled at how, if we created the EPS file (internally used for graphing) from the Python Command Line, it actually wrote (updated) the EPS files. However, when we transfered those OS commands to our program, the damn EPS file would *not* update what so ever. After losing hope that we would find the source of the error we tried many many hacks at this. As usual, hacks do not work. We wrote emails to the PYX authors and yet got no response. Looking through the PYX

[2]this happened about 2 weeks before the demo, so it was well into the semester as to change most of our work.

documentation *very very very carefully* we noticed the damn bastards had swept under the rug the fact that once you write an EPS file with their program, it is *finalized and non updatable whatsoever.* We started messing around with other ways to pass the data into PYX (since the file inputs were cached, and thus not updated on command) until we finally got Python lists to do the trick. Funny how things turn out, this method not only fixed our updating error, but it is *far* more efficient and cleaner to use.

For problem #3 we have no choice but accept the fact that PYX is dumb at times. We had two options for handling this problem. Either we could let the user know that when all the values were equal to each other the graph will come out as a big cube (representing the data all being equal) or manually changing the gprof parsed data by an insignificant amount such that no two adjacent values are equal. Now, the whole purpose of X3Profiler is to provide a graphical representation of gprof, so the latter option is really no option at all. No matter how insignificant the data change might be, it will invalidate the purpose of gprof. So, while it certainly is an uglier output, we will swallow this one and move on.

What can I say, Python doesn't like me (and I don't like Python) but most importantly, it did not like QStrings for some reason. We believe that Qt did some changes to it but the SIP files (SIP is a binding library between QT and Python) were not updated accordingly. This caused some weird errors that prevented our project from saving its internal state for quite a while. So, once we figured out that QStrings were to blame, we simply removed them and replaced them with an equivalent, more Python friendly Strings.

Problem # 4 is somewhat related to Problem #3, yet it merits special attention here. First of all, QT released and update on their software about 3 weeks ago (as of this writing.) We had Robert Rypka download this new version and play with it, just so we would know this update would break anything in our code. Well, it did. We lurked around several newsgroups and forums for the meaning of life, instead we found what the cause for the error was. The talented Mr. Rypka made a quick script that will fix this error and get our project back to working status on the latest QT release. (Note, this might come in handy for you when trying out our program since if you didnt already have QT in your system, you will probably end up with the new version. We do hope this fix works for you as well.)

Problem # 5 is more of a "testing ridiculous inputs" to ensure our program does not fail when correct, yet unreasonable, input is supplied. Well, from all we have seen, gprof text files are usually, *at the most*, roughly 40KB of data in size... so, I do not believe we will be seeing any gprof output file that even comes close to the 1 GB of data in size limit.

1.5 Other Types of Tests

The following is a brief summary of other types of tests and how they were done in X3Profiler:

Benchmark Test This really does not apply to our project as there is no other product similar to ours to base the Benchmark Test off. As such, X3Profiler does not take this test into consideration.

Configuration Testing Our product should be completely portable. Since it is written in Python, it should be portable to Windows, Linux, Mac's, etc. Now, given that we need

gprof files, we really aimed our product at anything that is Unix based. Thus far, our product performs exactly the same on 32 and 64 bit architectures as well as in Linux (specifically Fedora Core 3 and Gentoo), Windows XP (through Cygwin) and Mac OS X (10.3)

Functional Test Through the manual testing phase (see section 1.3) we ensured that all our use cases are fulfilled according to the Use Case Document created last semester.

Installation Test This test was actually performed at the beginning of the construction phase. While Python is indeed portable, a lot of bad python coders are out there. A lot of the modules written for Python break its portability by requiring specific operating system files or settings. This actually came as a shock for all the group members when we attempted to begin the construction phase of our project. Thus, we greatly examined which library dependencies we had in our project that could be dropped. In the end we got it down to just 2 main library dependencies. The first dependency is from PYX and the second dependency is from QT and SIP. Both of these should be fairly stable as to not pose serious problems for future installations.

Load Test For our application there really is no need for a Load test. Only one user will be using this application at a time (there is no networking involved whatsoever.)

Performance Test We never established a target maximum time for graphing and displaying a graph to the user, and as such we really can't strictly enforce one now. We can, however, user common sense for this test. If it takes too long (say about 20 seconds, which is very subjective) to accomplish this task, then X3Profiler will annoy its users. Overall, all of our graphs were generated and displayed within 2 seconds of being requested by the user. We estimate that 2 seconds is a fair amount of time for producing the desired output and thus consider our program as passing the Performance Test.

Stress Test Given the scenario in which most computer programmers work in (compiling possibly large pieces of software, running debuggers, browsing the internet, playing pirated music on iTunes, etc, etc) we have mostly restrained ourselves to consuming the Operating System's resources as much as possible by running other applications in addition to X3Profiler. Even under these conditions, our program managed to deliver acceptable results and produce the right output.