Experiences from Implementing and Using the Communication Link

Rishipal Bangarh

Astt. Prof., DAV College, Pehowa –Kurukshetra (India), Pin. No. -136128

Abstract – An open source project, with an ever changing group of developers, each with their own goals for the software, is by necessity different from a traditional software project. The major effects of being open source for the system have been more varied testing, more people doing debugging, and a need to keep the source code simple. WAP is being used all around the world, and implemented on many phones that only work in certain parts of the world.

----------♦----------

INTRODUCTION

Part 1: Discussion of what was done right and what was done wrong in the project, with regard to architecture, implementation, and project management.

Part 2: Discussion of how the open source development model has affected the project.

Part 3: Benchmarks on how the Communication Link performas at various load levels, and how it recovers from crashing wap and bearer boxes. Experiment designs are explained and results presented and discussed.

Part 4: Discussion of feedback from people using the Communication Link.

SUBJECTIVE EVALUATION

In this section I try to evaluate the system and describe things we've done right and things we've done badly.

It has not, however, made it easy to predict development speed, since at any time it may become necessary to throw aside the current development task and fix a customer problem.

In hindsight, the intense pressure for development speed was probably too intense, and resulted in slower development speed. Although some amount of pressure is good for getting people to work faster, and this resulted in overly optimistic time schedules and when they slipped, in further stress. From a software engineering and management point of view, the only redeeming feature of system's management is that the software made it to the market sufficiently early and with sufficient quality in order to succeed.

Building an open source development community around system has proved to be a much harder task. Partly this is because system is of interest to a fairly small group of people, but mostly it is because specially at the beginning the development discussions were not open.

The software development process itself has been loosely based on the spiral model although adapted to an open source development model, with rather fuzzy goals for each iteration. In short, the philosophy has been to get at least something working, so that people can try it out and even use it in production, and then improve and possibly rewrite it to make it better. Unlike many projects with this approach, the system has actually spent much time on the rewriting: code gets rewritten once it gets too buggy or it fits too badly with the parts around it that have changed. We have tried to keep the general architecture and internal interfaces clean, and thus rewrites have mostly been local. An excellent example of this is our HTTP implementation: the first one was made quickly, and served well for almost a year, and once its bugs and limitations in speed and features became problematic, it was rewritten completely from scratch without affecting the code calling more than by trivial calling convention changes.

A small, but very important things we did correctly was to set up a 'nag' script: a simple script to compile the current version, directly from the version control system, and mail the developers any error and warning messages. In principle, this script does what every developer should do, but it is hard to force developers to use a particular set of

compilation options, and even if they are willing, it is easy to forget one. The script helps by doing it automatically for all developers, and by doing it systematically every night. Additionally, when the script was run on multiple platforms, every night, it helped find several portability problems.

Later, we added some automatic test cases, which can be run by each developer. Even though the tests are simple, they do check for all the basic features of system, and make sure that a change won't break those. As time goes by, we add more tests, making it easier to catch more and more mistakes.

We recommend the nightly automatic compilation test and the automatic testing for all projects. Like most open source projects, we have been using a bug tracking system that anyone can browse. Our use of it has been unsystematic, though, with most bugs reported via email on the development mailing list. This has, at times, caused bugs to be ignored or forgotten. As system gains users, bug tracking will have to become more systematic.

Quality control in general has received rather little attenation from system. Except for the simple automatic test suite described above, the general approach of the developers has been to make their code or changes available, via the version control system, as soon as possible, so that others can participate in the testing. This is partly good, because the developers do not even have access to all mobile devices to do a complete test, and partly bad, because those areas of system that are hard to test or require specialized hardware, such as the SMS center protocol implementations, have been tested fairly lightly. On the whole, things have worked out, though.

EFFECTS OF CHOOSING TO BE OPEN SOURCE

An open source project, with an ever changing group of developers, each with their own goals for the software, is by necessity different from a traditional software project. The major effects of being open source for the system have been more varied testing, more people doing debugging, and a need to keep the source code simple. WAP is being used all around the world, and implemented on many phones that only work in certain parts of the world. Thus, for system to be compatible with all phones, it needs to be tested by people around the world, and in a traditional software project this would be quite hard to do. As an open source project, system has users from around the world, and they have helped in testing against almost all WAP capable phones in the world. Even though the testing is informal, i.e., there is no specific set of tests run by the users for each new link version, it is quite effective: as soon as a new and incompatible phone becomes available, or if the developers break system for some phone, the development mailing list gets bug reports.

This informal and distributed approach to testing has been applied to most parts of system development. The assumption is that if we have enough users, with different usage patterns, all or most code paths are exercised and if there are problems, we will hear about it. This, of course, flies in the face of conventional software engineering, but seems to work for us as it does for many open source projects.

The distributed approach also applies to debugging. One of the popular slogans for open source development is "when you have enough eyes, all bugs are shallow" . This does not mean that all bugs are easy to solve, but if there is an urgent problem with system, there will usually be many people working on finding it. They all work independently, but communicate about theirs findings and share theories. The end result is that the process of finding a particular bug is sped up significantly compared to having only one or two people working on it.

With many people working on same parts of the code together, communicating only over e-mail, it is important for the source code and program structure to be simple, so that everyone can understand it and so that fewer mistakes are made because of, for example, complicated interfaces or arcane programming tricks. Those parts that are complicated or tricky also tend cause more questions and more bugs.

The major impact of being open source, however, is more time spent communicating over e-mail. In a traditional project, much information is shared only orally, but since e-mail is the only common communication medium for the system, more time is spent reading and writing e-mail. On the other hand, much less time is spent sitting in meetings, and on the average the communication cost is probably about the same for system as it would be if the project wasn't open source.

REFERENCES

1. Georgiev, T., Georgieva E., Smrikarov, A., (2004). M-Learning – a New Stage of E-Learning, International Conference on Computer Systems and Technologies – CompSysTech'2004.

2. Giunta, G., (2002). Final Report on: Student Use of Mobile Learning in Italy. Retrieved 14, January 2005 from:

3. http://learning.ericsson.net/mlearning2/project_one /student_use_year_2_roma_tre.doc

4. Grumet, A. (2000). Adding Wireless Users To Your Web Service. Retrieved 14, January 2005 from: http://rhea.redhat.com/asj/wireless/

5. GSM World - The World Wide Web Site of The GSM Association, (2004) GSM Facts and Figures. Retrieved 13, December, 2004 from: http://www.gsmworld/news/statistics/pdf/gsma_stats_q2_04.pdf

6. Helic, D., Maurer, H., Scerbakov, N., (2002). Discussion Forums As Learning Resources In Web Based Education. Retrieved 27, April, 2004 from: http://coronet.iicm.edu/denis/pubs/ijca2004.pdf

7. Henke, H. (2005). Evaluating Web-based Instruction Design, Retrieved 20, May, 2005 from : http://scis.nova.edu/~henkeh/story1.htm

8. Hill, T.R., (2002). Leveraging Mobile Technology for m-Learning: 3rd Generation Threaded Discussions, Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03).

9. Homan, S., Wood, K., (2003). Taming the Mega-Lecture: Wireless Quizzing. Retrieved 14, January 2005 from: http://www.campus-technology.com/campusmobility/article.asp?id=8251