Understanding the Requirements for Open Source Software Development

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
Wscacchi@ics.uci.edu

http://www.ics.uci.edu/~wscacchi/Presentations/OSS-Requirements
Overview

• Research methodology
• Community characteristics
• Software Requirements process
• Open source processes for Requirements
• Software Informalisms
• Implications
• Conclusions
Research methodology

• Prior empirical (case) studies of Open Source Software Development (OSSD) Projects
  – Reis and Fortes, 2002, Mozilla Web browser
  – Schach et al., 2002; Holt et al., 2000, Linux Kernel
  – Koch and Schneider 2001; German 2002, GNOME User Interface
  – Jorgensen, 2001, FreeBSD operating system
  – Garg et al., 2002, OSSD (“progressive open source”) within HP
Research methodology

• Individual case studies: significant details, but limited (and premature) generalization, little/no comparative analysis
  – Halloran and Scherlis, 2002, comparative study of software tools and code volume in eleven OSSD projects, all in one domain (Internet infrastructure)

• No studies that examine multiple OSSD projects in multiple domains
  – Such studies would offer higher degree of comparative analyses and generalization of results
Research methodology

• Comparative case studies
  – Multiple open software development projects
    • Across four communities
      – Two research oriented
      – Two development oriented

• Qualitative ("grounded theory") techniques

• Analyzing and modeling
  – development processes
  – work practices
  – community structures
Community characteristics

- According to Steve Ballmer (CEO, Microsoft)
  - "We have to compete with free software, on value, but in a smart way. We cannot price at zero, so we need to justify our posture and pricing. Linux isn't going to go away--our job is to provide a better product in the marketplace."
  - "Linux is not about free software, it is about community”(emphasis added).
  - London, 24 September 2002, speaking on MS, its “Most Valued Professionals” (MVPs), and “shared source” vs. “open source”
Community characteristics

• Development oriented domains
  – Networked computer games
  – Internet infrastructure

• Research oriented domains
  – Astrophysics/deep space imaging
  – Academic software design
Software Requirements process

• Classic Requirements Engineering Process
  – Elicitation
  – Analysis
  – Specification and modeling
  – Validation
  – Communicating and managing
Open source processes for Requirements

- *Post-hoc* assertion of requirements+design
- Reading, sense-making, accountability
- Continually emerging webs of discourse
- Condensing and hardening discourse
- Global access to discourse
Open source processes for Requirements

- OSS Requirements are
  - not explicit
  - not formal

- QED, OSS Requirements embedded within “informalisms”

- Example OSS informalisms follow
Benefits of Qt3?
by Matt Perry on Friday July 27, @09:22AM

What are the benefits of moving to Qt3?

[Reply To This | View ]

* Re: Benefits of Qt3?
  by Justin on Friday July 27, @09:41AM
  - Support for Atrias and Hewitt
  - Rich Text classes
  - Database support
  - Component model
  - No more cut/paste problems (but only between Qt5 apps)

One of the most complained about aspects of X is the dmx clipboard, so getting KDE based on Qt5 will solve a lot of headaches. But this is from a user perspective.

From a developer perspective, KDB-DB is going to utilize Qt3's database support, and this can't happen until they make the switch. KWord currently uses a backported nicetext for use with Qt2. So you can see that there is a drive/need in KDE to use the new Qt3 features.

[Reply To This | View ]

* Re: Benefits of Qt3?
  by Mike on Friday July 27, @12:04PM
  - What is the purpose of database support in a "widget toolkit"? Isn't this just like placing 'TZ/EF' support in /etc/passwd or another similarly unrelated place?

[Reply To This | View ]

* Re: Benefits of Qt3?
  by Aaron J. Seigo on Friday July 27, @12:36PM

  there is often a need to access data from a database and display it in a GUI, or vice versa, in those cases having a db API that abstracts the details of the actual data access away (connecting, sending queries, retrieving results, details specific to a given db implementation, etc) that works nicely with your widgets (even so far as to make the widgets aware of the database) is very very nice.

  making such things simple and convenient opens the door to making more applications database aware (e.g. financial packages, email apps, contact information systems)

[Reply To This | View ]
Ideas

Here are a few ideas of tools, classes and projects that you could start. More are forthcoming.

Runtime

We need a verifier that can be run on an executable (assembly) and tells whether the metadata for the executable is correct or not. It should report any anomalies.

For a list of anomalies in assemblies, check the various assertions that are described on the ECMA documentation.

This will help test our generated executables and can also be used as an external verifier.

Classes

- TODO: jxta, The JXTA Peer to Peer Foundation
  - Implement a JXTA protocol implementation: http://www.jxta.org
- TODO: mail, Mail API
  - Implement a Mail API, similar to Camel or JavaMail (Camel has significant architecture features that are required on a real mailer).
  - You can check the current Camel implementation.
    - Such an implementation could be used both with Microsoft .NET and Mono.
- TODO: multimedia
  - Interfacing to Multimedia systems. You might want to look into the Quicktime API. I know Vladimir has researched the problem before.
- TODO: gtk+, Gtik+ wrappers for Mono and .NET
  - Wrap the Gtik++ API. This is simple and can be done on Windows as Gtik++ 2.0 works on Windows.
  - This work can also be used on Windows and will enable developers on Windows to use some of Gtik++'s advanced features.
  - The idea is to wrap the Gtik++ API and allow us to build GUI applications using Gtik++ and in the future either .NET-based libraries from Mono (GSI, Gtk#.ML).
  - There is extensive knowledge on wrapping Gtik++ in other languages. Gtik++ has been done this for Perl, Python, Java, Scheme, Haskell and other languages in the past.

Projects

- Implement an xmlStorageSystem for the CLI: http://www.soapware.org/xmlStorageSystem
- TODO: guava, Java compiler for .NET
  - You could take one of the existing Java compilers (Guava comes to mind as it is so nice) and modify it to generate .NET code rather than Java byte codes.
  - This should be a pretty straightforward task. Guava has the advantage of being written in C++ and it could be compiled with the Microsoft Managed C++ compiler and produce a .NET executable with it.
The Chandra Automatic Data Processing Infrastructure

David Flummer and Sreenatha Subramanian
Harvard-Smithsonian Center for Astrophysics, 60 Garden St. MS-81, Cambridge, MA 02138

Abstract:
The requirements for processing Chandra telemetry are very involved and complex. To maximize efficiency, the infrastructure for processing telemetry has been automated such that all stages of processing will be initiated without operator intervention once a telemetry file is sent to the processing input directory. To maximize flexibility, the processing infrastructure is configured via an ASCII registry. This paper discusses the major components of the Automatic Processing infrastructure including our use of the STScI OPUS system. It describes how the registry is used to control and coordinate the automatic processing.

1. Introduction

Chandra data are processed, archived, and distributed by the Chandra X-ray Center (CXC). Standard Data Processing is accomplished by dozens of "pipelines" designed to process specific instrument data and/or generate a particular data product. Pipelines are organized into levels and generally require as input the output products from earlier levels. Some pipelines process data by observation while others process according to a set time interval or other criteria. Thus, the processing requirements and pipeline data dependencies are very complex. This complexity is captured in an ASCII processing registry which contains information about every data product and pipeline. The Automatic Processing system (APS) polls its input directories for raw telemetry and ephemeral data, pre-processes the telemetry, kicks off the processing pipelines at the appropriate times, provides the required input, and archives the output data products.

2. CXC Pipelines

A CXC pipeline is defined by an ASCII profile template that contains a list of tools to run and the associated run-time parameters (e.g., input/output directory, root-names, etc.). When a pipeline is ready to run, a pipeline run-time profile is generated by the profile builder tool, pbuilder. The run-time profile is executed by the Pipeline Controller, pctr. The pipeline profiles and pctr support conditional execution of tools, branching and converging of threads, and log file output containing the profile, list of run-time tools, arguments, run status, parameter files, and run-time output. This process is summarized in Figure 1.

![Figure 1: The CXC Pipeline Processing Mechanism](image-url)
4. Standard Pipeline Processing Threads

Figure 2 represents the series of pipelines that are run to process the Chandra data. Each circle represents a different pipeline (or related set of pipelines). Level 0 processing (De-commutation) will produce several data products that correspond to the different spacecraft components. Data from the various components of the spacecraft will follow different threads through the system. The arrows represent the flow of data as the output products of one pipeline are used as inputs to a pipe (or pipes) in the next level. Some pipelines are run on arbitrary time boundaries (as data are available) and others must be run on time boundaries based on observation interval start and stop times (which are determined in the level 0 pipe, OBI_DET).

5. Pipeline Processing Registry

The complete pipeline processing requirements for Chandra are very complex with many inter-dependencies (as can be seen in Figure 2). In order to run the pipelines efficiently in a flexible and automated fashion, we configure the Automatic Processing system with a pipeline processing registry. We first register all the Chandra input and output data products. We can then capture the processing requirements and inter-dependencies by registering all the pipelines. Data products are registered with a File_ID, file name convention (using regular expressions), method for extracting start/stop times, and archive ingest keywords (detector, level, etc.). Pipelines are registered with a Pipe_ID, pipeline profile name, pipeline arguments, kickoff criteria (detector in focal plane, gratings mosaic, etc.), input and output data products (by File_ID), and method for generating the "root" part of output file names.
Open source processes for Requirements

- Elicitation
- Analysis
- Specification and modeling
- Validation
- Communicating and managing

- *Post-hoc* assertion
- Reading, sense-making, accountability
- Continually emerging webs of discourse
- Condensing and hardening discourse
- Global access to discourse
Software Informalisms

• Community communications
  – Threaded discussion forums
  – Email (list servers)
  – Newsgroups
  – IRChat/Instant messages
  – Community digests ("Kernel Cousins")
Software Informalisms

- *Scenarios of Usage* as linked Web pages
Software Informalisms

• How-To guides, To-Do lists, FAQs
• Traditional software user documentation
  – Unix/Linux man pages
• External publications
  – trade articles
  – scholarly research papers
  – books (cf. O’Reilly Books)
Software Informalisms

• Open Software Web Sites
  – Community Web sites
  – Community Software Web sites
  – Project Web sites
  – Source code Webs/Directories
AcmeStudio
A Graphical Design Environment for Acme

AcmeStudio is a 32-bit Windows 95/NT application for graphically editing architectural descriptions represented in the Acme ADL (Architecture Description Language). An Acme description can be opened in AcmeStudio, edited graphically and saved back into Acme.

AcmeStudio includes a mechanism to create different "diagram styles" in which Acme Systems can be viewed. Using a diagram style you can define new visualizations for the types defined in an Family, and define variations on the basic visualization based on property values.

AcmeStudio now includes built-in support for the queueing network theory based performance analysis as well as simple general support for user-defined external tools.

Downloading and Using AcmeStudio

As part of the download process, we ask that you register with us:

Register and download AcmeStudio

Installation Notes:

The AcmeStudio setup program does the following to your system: Installs the executables, and DLLs in the installation directory you specify during setup. Creates registry entries for the "*.acme" file type to associate Acme description files with AcmeStudio under HKEY_CLASSES_ROOT. Creates registry entries that hold the installation directory and other information in HKEY_LOCAL_MACHINE\SOFTWARE\Acme\AcmeStudio.

NOTE: AcmeStudio will now run under Windows 95/98. However, it has been much more thoroughly tested under Windows NT.
Software Informalisms

• Software bug reports
  – Ad hoc report Web
  – Bugzilla (database tracking)

• Issue tracking
  – Issuezilla
Bug: dmextract

- Bug: Crashes when extracting a PHA, while using a stack of files for background extraction [6155]

- Bug (fixed): A memory corruption causes the tool to crash when creating a long HISTORY string [6205]

- Caveat: BACKSCAL areas for complicated regions may have small (of order 1-2 percent) errors, as they are calculated using an approximate algorithm. CIAO 2.2 is improved relative to CIAO 2.1, but still not perfect.

Last modified: 14 November 2001
Software Informalisms

• Software extension mechanisms
  – Inter-application scripting
    • Csh, Perl, Python, Tcl, scripting
    • Pipelines (cf. CXCDS)
  – Intra-application scripting (e.g., UnrealScript)
  – Plug-in architectures
    • Apache server architecture
Software Informalisms

• Open source software licenses
  – GNU Public License (GPL)
  – Lesser/Library GPL (LGPL)
  – Artistic License
  – Mozilla Public License (MPL)
  – SUN Public License (SPL)
  – and 25 more (http://opensource.org)
  – “Creative Commons” Project at Stanford Law School developing public license framework
Implications

- Software informalisms are the *media* of software requirements
- Software informalisms are the *subject* of software requirements
- OSS Requirements are *implied activities or capabilities*
- *(Re)reading and reviewing* informalisms is a prerequisite to *writing* open software
Implications

• Developing open software requirements is a community building process
  – not just a technical development process
  – open source peer reviewing creates a community of peers

• OSSD processes often iterate daily versus infrequent singular (milestone) SLC events
  – frequent, rapid cycle time (easier to improve) vs. infrequent, slow cycle time (hard to improve)
Implications

• Determining the quality of open software requirements:
  – not targeted to consistency, completeness, correctness
  – instead focusing attention to community building, freedom of expression, ease of informalism navigation (traceability), implicit vs. explicit informalism structuring
Conclusions

• Developing open software requirements is different than requirements engineering
  – not better, not worse, but different and new
  – more social, more accessible, more convivial

• Open source software systems don’t need and probably won’t benefit from classic software requirements engineering.
Conclusions

• Need to *integrate* OSSD with SE
  – development infrastructure (tools and environments)
  – development processes
  – developer community
  – across multiple domains
    • Scientific research
    • Commercial development
Conclusions

• People use OSS development tools to create, update, distribute, or browse OSS informalisms

• OSSD tool taxonomy:
  – Seven level hierarchy; more than 40 tool types
Acknowledgements

• Project collaborators:
  – Mark Ackerman, UMichigan,
  – Margaret Ellliot, Ph.D., Mark Bergman, Xiaobin Li, UCI-ISR
  – Julia Watson, The Ohio State University

• Funding support:
  – National Science Foundation, IIS#-0083075, ITR#-#0205679
  – Defense Acquisition University, N487650-27803
References

- W. Scacchi, Exploring Open Source System Acquisition Processes and Architectures, Final Report, June 2002,  