OHJ1860 Open Source Software

The production and release process in F/OSS

Jaco Geldenhuys, Tue 28 Mar
Overview

• P/CSS production & release process
  – What is the P/CSS production & release process? ........................................ 2
  – Meta-processes for software ................................................................. 3
  – Processes for software ........................................................................... 5

• F/OSS production & release process
  – Processes for F/OSS .............................................................................. 6
  – Raymond ................................................................................................. 7
  – F/OSS v. Agile .......................................................................................... 10
  – Singh/Fan/Tan .......................................................................................... 11
  – Erenkrantz ................................................................................................ 12

• Some explorations with moodle
  – Contribution quanta ............................................................................... 15
  – Activity time ............................................................................................ 18
  – File activity .............................................................................................. 20

• Summary .................................................................................................... 21

• Questions for discussion .......................................................................... 22
What is the P/CSS production & release process?

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. (*IEEE Standard Glossary of Software Engineering Terminology, IEEE std 610.12-1990*)

- Software engineering = knowledge, tools, and methods for defining requirements, design, construction, testing, and maintainence of software.

- Also skills from computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics, and systems engineering.
Meta-processes for software

ISO/IEC 12207
Software Life Cycle Processes, Aug’95

ISO 12207

IEEE Stds

IEEE/EAI 12207.0-1996
IEEE/EAI 12207.1-1996
IEEE/EAI 12207.2-1996

DOD-STD-2167A
Defense System Software Development, Feb’88

DOD-STD-7935A
DoD Automated Information Systems (AIS) Documentation Standards, Oct’88

MIL-STD-498
Software Development and Documentation, Dec’94

Software Life Cycle Processes, Software Development, Sep’95

IEEE/EAI 12207.0-1996
Software Life Cycle Processes, Apr’98

IEEE/EAI 12207.1-1996
IEEE/EAI 12207.2-1996

Meta-processes for software

Life cycle meta-processes
- ISO 12207
- IEEE Stds
- 2167A
- 498
- 016
- 12207
- 7935A

Maturity meta-processes
- ISO 15504
- CMMI
- Six Sigma

Quality assurance meta-processes
- ISO 9001
- ISO 90003

4/22
Processes for software

- Waterfall
  - Requirements
  - Design
  - Implementation
  - Verification
  - Maintenance
  - BDUF
  - SSADM
  - SDM2

- Iterative
  - Planning
  - Requirements Analysis & Design
  - Implementation
  - Testing
  - Evaluation
  - Deployment
  - Initial planning
  - Spiral
  - Agile
    - XP
    - Scrum
    - DSDM
  - RAD
  - ASD
  - RUP (Rational)
  - UML
  - Cleanroom
  - RAISE
  - Z
  - B-Method
  - VDM
  - Formal methods
  - Cleanroom
  - SDM2
  - SSADM

- Formal methods
  - Cleanroom
  - RAISE
  - Z
  - VDM
  - B-Method
  - Formal methods
Processes for F/OSS

- Free software has no market pressure ⇒ less need for (centralized) structured process to drive development

- So what is the F/OSS process? (We don’t necessarily want to formalize it; merely study it from an anthropological perspective → descriptive instead of prescriptive)
Raymond

In “The Cathedral and the Bazaar” (1997-2000) Raymond states a number of lessons which might give us some insight into F/OSS processes:

1. Every good work of software starts by scratching a developer’s personal itch.
   ⇒ Early requirements are more personal

2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
   ⇒ Process is iterative; refines earlier projects

3. Plan to throw one away; you will, anyhow.
   ⇒ Process is repetative

4. If you have the right attitude, interesting problems will find you.
   Context: Raymond’s project had escalated.
   ⇒ Process is evolutionary
5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

⇒ Raises questions about central control & continuity of control

6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.


⇒ Process is short-circuited; early release means that testing begins early; cf. agile software development
8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
   ⇒ Large, diverse developer pool possible
   Raymond calls 8. “Linus’s Law: Debugging is parallelizable”
   ⇒ Process works in parallel, unsynchronized fashion

12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
   ⇒ Design and analysis may change late in the process
F/OSS v. Agile

Similarities

• Process is iterative
• Control is relatively informal
• Process is parallel
• Early releases ⇒ early testing
• Early releases ⇒ progress measured by working software

Differences

• Iteration in F/OSS ⇒ early release encourages participation
• Iteration in Agile ⇒ limited release minimizes risks
• No domain specialists
• No documents/deadlines required for F/OSS
• Market (or separate user community) is not the source of the problem

Findings:

- Top 20% of most contributing developers contribute $\sim 80\%$ of code
- Top 30% of most contributing developers contribute $\sim 81\%$ of messages
- Projects with clearly defined core group writing most of the code are more efficient at project progress
- Highly participative peripheral group in communication positively affects progress
- Negative factors for progress: (i) lack of a core developer group, (ii) dormant peripheral group, (iii) low activity in threaded communications, (iv) over-dependence on internal community for beta testing.

<table>
<thead>
<tr>
<th>Release authority</th>
<th>Linux</th>
<th>Subversion</th>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designator</td>
<td></td>
<td>Subset of committers</td>
<td>Committer</td>
</tr>
<tr>
<td>Stable/unstable</td>
<td></td>
<td>Pre-1.0</td>
<td>Stable/unstable</td>
</tr>
<tr>
<td>Release candidates</td>
<td></td>
<td>Regression</td>
<td>Real-world, automated</td>
</tr>
<tr>
<td>Release manager</td>
<td></td>
<td>Release manager</td>
<td>Consensus</td>
</tr>
<tr>
<td>Mirrors</td>
<td></td>
<td>Single site</td>
<td>Mirrors</td>
</tr>
<tr>
<td>No official contributions</td>
<td></td>
<td>User contributed</td>
<td>Committer contributed</td>
</tr>
</tbody>
</table>

- Release manager ← maintainer ← public
- Versions numbered $x.y.z$, even $y \Rightarrow$ stable
- Linus Torvalds usually in charge of unstable branch, decides when release is stable enough to hand to release manager
- Unstable releases tested by users
- Release manager controls decision to release
- Released as compressed tar-files of source
## Erenkrantz

<table>
<thead>
<tr>
<th>Release authority</th>
<th>Linux</th>
<th>Subversion</th>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>Designator</td>
<td>Subset of committers</td>
<td>Committer</td>
</tr>
<tr>
<td>Pre-release testing</td>
<td>Stable/unstable</td>
<td>Pre-1.0</td>
<td>Stable/unstable</td>
</tr>
<tr>
<td>Approval of release</td>
<td>Release candidates</td>
<td>Regression</td>
<td>Real-world, automated</td>
</tr>
<tr>
<td>Distribution</td>
<td>Release manager</td>
<td>Release manager</td>
<td>Consensus</td>
</tr>
<tr>
<td>Formats</td>
<td>Mirrors</td>
<td>Single site</td>
<td>Mirrors</td>
</tr>
<tr>
<td></td>
<td>No official contributions</td>
<td>User contributed</td>
<td>Committer contributed</td>
</tr>
</tbody>
</table>

- Infrastructure + core developers funded by CollabNet
- Only available in alpha releases (2007: Version 1.4.3)
- Automated test suite
- Core developers use issue tracking system; some issues must be addressed to reach next release milestone
- Released as compressed tar-files of source; users are encouraged to contribute packages (Debian, RPM, Windows, ...)

13/22
<table>
<thead>
<tr>
<th>Release authority</th>
<th>Linux</th>
<th>Subversion</th>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designator</td>
<td>Subset of committers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stable/unstable</td>
<td>Pre-1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release candidates</td>
<td>Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release manager</td>
<td>Release manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mirrors</td>
<td>Single site</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No official contributions</td>
<td>User contributed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Project organized as meritocracy with three groups: management committee ↔ committer ↔ developer
- Release manager is volunteer from management committee
- Version numbering same as Linux
- “Stable” releases first run on apache.org website
- Release manager must decide what goes into release, but rest of group must approve release
moodle: contribution quanta

Number of submission days

Number of files changed

1127

1638
moodle: activity time
Summary

- Difficult to classify processes: due to diverse nature of F/OSS
- F/OSS show some similarities with Agile development
- Process can be said to be unstructured, and more important question is the patterns of developer behaviour:
  - Who submits? Large or small group
  - Who participates in the discussion forums?
  - How much discussion is involved before changes are made?
Questions for discussion

• How true are the following statements?
  1. F/OSS is free (as in “free beer”) ✓ most users pay nothing ✗ lost productivity
  2. F/OSS is free (as in “free speech”) ✓ web is democratic ✗ ...
  3. F/OSS is high-quality ✓ by example ✗ often not the case
  4. F/OSS is efficient faster than P/CSS?
  5. F/OSS model is successful
     ✓ by example: Linux, GCC, KDE, GNOME, Apache
     ✗ like ESP: many projects fail invisibly and spectacularly

• Can F/OSS model be adapted for P/CSS?
• Can a meta-process like ISO 9001 be applied to F/OSS?