Lessons Learned with Time Based Releases for the EFL

FOSDEM 2015

Stefan Schmidt
Samsung Open Source Group
s.schmidt@samsung.com

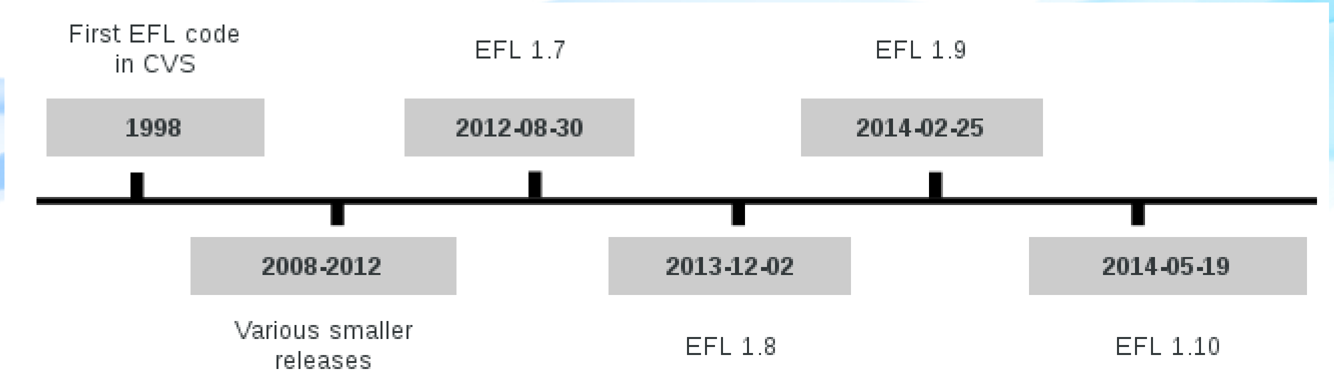
Agenda

- Release History
- Current State
- Lesson 1: Build Trust into the Code
- Lesson 2: Automate to Keep You Sane
- Lesson 3: Social Changes are Harder than Technical On es
- Lesson 4: Keep Refining the Process and Results
- Summary

Release History

Release History

- The Enlightenment project started in 1997
- Some early releases until E 0.16
- A 12 years gap from first code to release E 0.17
- Building a whole new set of libraries for it but no releases for them either for a long time
- More release driven but still feature based until 1.9



Current State

What Changed?

- Most Linux distributions only package released software
- Commercial parties also asked for releases
- With the pressure also the understanding grow that releases would help the project forward

Release Cycle

- Only applies for the core libraries and not the window manager
- 3 month release cycle (12 weeks to be exact)
- Starts with a 8 weeks merge window
- 4 weeks final stabilization with alpha and beta releases

Schedule

- Schedule is agreed up and set in advance
- Schedule is mostly stable these days besides changes due to holidays, etc
- Slips are only allowed for serious problems
- Normal bugfixes will get delivered as stable updates

Switch to Time Based Releases

Started with 1.9 in December 2013

- 1.9 was released 1 day late (February 2014)
- 1.10 was released 7 days late (May 2014)
  - At that point the change for 3 weeks stabilization started
- 1.11 was released 2 days late (August 2014)
- 1.12 was released released on time (November 2014)
  - After this release we switched to one merge window
- 1.13 is planned for 9th of February

Lesson 1: Build Trust into the Code

Code Complexity

- The biggest roadblock is to build trust in the code base
- In the Enlightenment project the two main libraries alone are having around 940.000 lines of code
- The window manager has another 270.000 lines
- You need a variation of tools that will help you to build up this trust
- SCM, automated builds and maybe continuous integration, static analyzers just to name a few

Code Complexity

- Historically the Enlightenment project was happy to add configuration options
- Over the last two years we tried to reduce these options
- We also merged 11 different (small) libs into one EFL lib
- All of these actions had direct influence on how much different configurations we can test
- The reduced set of libraries also eased the release process

Lesson 2: Automate to Keep You Sane

Automated Builds

- Automated builds with Jenkins with quick builds for every push to master
- Building for x86, x86_64, x32 with gcc, clang and mingw
- Very useful as many developers only use one system for their work
- Many of the discovered problems are getting fixed instantly due to awareness and social pressure
- In the dark times before we had this setup builds have been broken for some configurations for months

Static Analyzer

- LLVM/scan-build: output quite noise and sadly a lot false positives
- Klocwork: the proprietary nature made it hard to share results
- Now using Coverity which is also proprietary but offers a free service for FOSS projects
- Especially Coverity uncovered quite some issues which have been overlooked during code review
- You still have to deal with false positives but the signal to noise ratio is way better

Lesson 3: Social Changes are Harder than Technical Ones

What Hold Releases Back?

- Most of the active developers at that time did not see a need for releases
- It was a project driven by volunteers which used the code straight out of SCM
- Most people do not like change in general
- Nobody stood up for taking care of the releases so it did not get done

General Social Changes

- Once we had IRC and email notification about broken builds social pressure grow to fix them
- Keep people in the loop and highlight positive changes (e.g. weekly QA newsletter)
- Most people have been ok with adapting to changes but you have to provide docs, guidance and time
- People follow the lead but would not have done it on their own. (e.g. backports of fixes to stable branches)

Ongoing Releases

- Easier once we came over a long time without releases
- Getting into the habit to release more often
- Core application now do releases a while after the libraries. Happened naturally.
- Invest time to make a sane and realistic schedule and fine tune.

Lesson 4: Keep Refining the Process and Results

Tweaks to the Schedule

- We started out with a more complicated release schedule
- 4 weeks merge window, 2 weeks stabilization, 4 weeks merge window and 2 final weeks of stabilization
- After 2 cycles we moved one stabilization week to the end as we needed more time for the final phase
- Three cycles later we switched to an easier schedule with 8 weeks development and 4 weeks stabilization

Tooling

- Automated NEWS file generation
- For this we introduced the @feature and @fix tags for commit messages
- Automated building, tarball generation, uploading,...
- ABI checker runs to spot ABI/API problems
- Documentation now gets generated and uploaded automatically for a new release

Summary

What Yoo Should Take Home

- Automation keeps the workload down and the project in good shape during the development phase
- Invest time in these tools instead of investing it in manual testing
- Keep on fine tuning the process
- Expect that people will only follow if you lead

Thank you.

We are hiring.

jobs@osg.samsung.com

References

- http://www.enlightenment.org
- http://build.enlightenment.org
- Imagery sources
- https://openclipart.org/detail/192629/gear-tools-by-ben-192629
- http://jenkins-ci.org/
- https://openclipart.org/detail/26498/present-blue-pack-by-minduka