COLLABORA

# Managing Client's Projects in Opensource and Being Profitable

## Making sustainable contributions to opensource

**Alvaro Soliverez**
**Project Manager**
**Collabora Ltd**

Open First

COLLABORA

# Introduction

## Matchmaking worlds with very different views

COLLABORA

## Customer Projects

- Project goal is aligned with a business goal
- Has a tightly defined budget
- Has a defined, and usually tight, timeline
- Has a set of acceptance criteria to fulfill
- Constrained by budget, time, quality, scope, and business goals

COLLABORA

## Community Project

- Project goal is aligned with a technical goal
- No defined budget
- Some projects will have timelines, others will “release when done”
- Fluid set of acceptance criteria, no a priori definition
- Constrained by quality, time availability of contributors, and technical relevance
- All projects are different, get to know them

COLLABORA

# A balancing act

- Keep an eye on the budget spreadsheet, and the other on the commit log
- Adapt our talking points to these different needs
- Overgeneralizing:
  - Community focus is on features and code quality, not budget
  - Customer focus is on deliverables and cost, not maintainers' demands
- Our job is to preserve the health of both our project and the community project
- It's a balancing act, but it pays off if done well

COLLABORA

## What we want

- Fulfill customer goals and contribute to open source at the same time
- Respect our motto and mission
  - “Open first”
  - “Accelerate the adoption of open source technologies, methodologies & philosophy”

COLLABORA

# What we want – cont.

- We want to contribute and get better at it
- If we make a profit, we can continue our contributions
- Be fair to our customers and the community

# “Open First”

- Public by default
- Always insist on working in the open. Demand good reasons to do otherwise
- Code repository in the open (reference to upstream for easy porting)
- Whenever possible, use original project bug trackers
- Caveat: we do always use an internal one for internal project tracking

COLLABORA

# “Accelerate the adoption of open source technologies, methodologies & philosophy”

- Doing our job well, we can demonstrate the power of open source to customers
- We work this way because we are convinced it’s the most productive and cost-effective way
- We can believe it’s so, but there’s no better proof than hard cold numbers
- Profit!

## Why help upstream?

- If done well, it makes big economic sense, for us and our customers
- We use it and sometimes maintain it too
- Other customers build products with it too
- Any patch going in improves the end result

## Some common hurdles

- Customer projects have tight deadlines and budgets
- Customers new to open source may not see the value in upstreaming
  - Fortunately, this happens less and less
- Some kind of education is required to show the value of open source and upstreaming

# Some common hurdles – cont.

- Contributing has to fit in the project plan
  - Upstreaming as an after-thought is a recipe for failure
- Harder to justify in short projects
  - Why would they care about long-term benefits?
- Or drive-by projects that will never touch a project again

## Some common hurdles – cont.

- For service-based companies, every project is a race to the bottom line
- We have to break even or we are out of business, there's no cash-cow product to rely on
- And without a big marketing budget, our reputation in the open source world is our best advertising

## How to prepare

- Assess the project
  - How much does it overlap with upstream projects?
  - Is it about new features or customization?
- Assess customer's attitude
  - Willing/able to contribute?
  - Undertands open source?
- Project relevance to us
  - Drive-by or strategic project?

COLLABORA

# Some questions at first

- Ask the customer:
  - What license to use?
  - Can we upstream the code?
  - Can we mention upstream whom we are working for?
  - Can we mention the project?
  - Can we discuss the specific use case?
  - Can we mention the hardware?
  - Most or all should be in the SOW
- Any “No” adds difficulty to the process

## Working with No

- “No” to upstreaming code
  - Worst situation, contribution to upstream will be minimal and on our own time
- “No” to mentioning customer, project, hardware
  - Upstream will lack context on your situation
  - Communication gets more difficult
- Basically, any No increases the burden

## Clarify value to customers

- It’s an ongoing task, not a single item in the plan
- Coordinating with upstream can result in tasks being done by other projects
  - Get extra features for free
- Improve long-term maintenance
- Better communication, better insight into the future and more influence in direction
- For single projects, it might still be difficult
- Easier when it’s recurrent work

# Clarify value to the team

- Even if not paid by customer, it may be cheaper to upstream on our own in the short-term
- Can it minimize development time/cost?
- Can it minimize support cost?
- Document what can not be upstreamed during the project but could be done later

## Project management

- Upstreaming is not just one task, is a process
- Try to secure budget for it
- Account for gains and expenses
  - Extra features, better support
  - Extra effort and time spent upstreaming

# Planning

- Upstream has to be taken into account from requirements definition to testing and bugfixing
- Allow for feedback times
- Allocate adequate people to the project

# Planning tactics

- Is this feature being worked by someone else?
- Can we join efforts?
- Can we delay part of it until upstreamed by someone else, and then add on top?
- Communicate with upstream and nearby projects
- Plan contingency if other projects don’t fulfill
- It’s a balancing act

## People allocation

- If possible, get people related to upstream
- Smooth communication is key
- Upstreaming is easier when it comes from an existing contributor
  - More experience, existing network
- If not, start building a relationship now!

# Planning tactics

- Be creative
- Look at the budget sheet and the calendar
  - That includes release schedules
- Negotiate and then negotiate some more:
  - “We work on X feature, they work on Y, and we both get X and Y at half the cost!”
- Keep the customer on the loop
  - After all, any tactic has a risk, and the customer must be aware of it

## Communication with upstream

- Code does the best talk
- Quality is important
- No “throwing code over the fence”
- Talk is mostly in technical terms. The PM should take a step back and let developers take lead here
- Context helps on reviews, provide as much as possible, within customer constraints
- Build genuine relationships, no pep talk

## Requirements definition

- Will this feature be accepted by upstream?
- If so, can we secure a maintainer to provide timely feedback?
- If not, can we define it in a way to minimize non-upstreamed patches?
  - Tailor it
  - Break it down
  - Account for increased maintenance costs

## Development

- Be open about dependencies
- Keep in touch with all participants and ask for feedback early
- Plan to have one first version “that works” and integrate a “correct” one later after upstreaming
- Plan for review and fix times

# Testing

- Consider your project and upstream testing requirements
  - Testing requirements might be steep for upstreaming, have to be planned early
- Leverage upstream CI testbed if it exists
- Integrate early
- Fix issues promptly

## Communication

- Keep a close eye on what upstream is doing
- Keep the customer on the loop for relevant features
- Be flexible with the plan, leverage the opportunity if there is momentum for a desired feature
  - “Nothing is more powerful than an idea whose time has come”
  - Use it to our advantage

## Communication

- Blog about what we are doing
- Telling we are working on x feature might spur interest from other parties
- No public relation stunts, be as transparent as possible

## Retrospective

- During the project retrospective, consider the upstreaming process
- What can be improved, what went wrong
- Go over the list of non-upstreamed patches
- Highlight the ones that are worth having upstream

# What if you can’t?

- What if we can’t upstream during the project?
- Keep a list of patches
- Whatever we do, try to do it early or patches will bitrot quickly
  - That is wasted effort for everyone, not good

# Tactics to finish upstreaming

- Is there a sequel to the finished project?
  - Can we include this effort in the new project?
- Internal company project for upstreaming
  - Slowly upstream the most relevant patches
- File and submit the patches upstream
  - The attention these will receive depends greatly on developers being available

# Tactics to finish upstreaming – cont.

- File and submit the patches upstream
  - The attention these will receive depends greatly on developers being available
  - This is bad, but still better than let them bitrot in a forgotten repository

## Examples

- Project based on Debian
  - Any fix in a package is submitted directly to Debian and then automatically fed back into the project from Debian repository
  - Upstream is part of the regular workflow, no added cost, just a bit extra time

# Examples – cont.

- Android on stock Linux
  - Waited for features done for ChromeOS
  - Additional fixes required changes in etnaviv, Mesa, AOSP
  - Discussed with all projects and all upstreams what was acceptable
  - Worked out schedule, split out work
  - Worked on our own part, supported other projects with feedback and testing
  - Big feature at 1/3 the cost

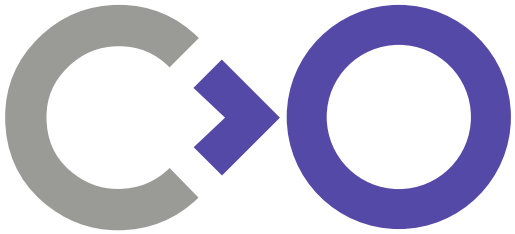
COLLABORA

# Experience pays off

- Upstreaming is second nature to us
- We can quickly point out issues and get to creative solutions
- We get contracted to upstream other companies' work
- Good karma for everyone

## Some Conclusions

- Communicate up, down, and sideways
- Leverage what other projects are doing
- Upstream is a process through the whole project
- Show value in upstreaming
- Avoid throwing code over the fence
- Keep track of patches pending to upstream



# Thank you!