July 23, 2020 – GUADEC2020

Bringing your favorite GNOME desktop app back from the dead

Jean-Francois Fortin Tam

>> Next talk will be in 10 minutes.

>> Testing, testing. This is just checking that, A, the microphone works. And B, that the video quality is okay.

>> It is.

>> This quality thing.

>> Both are good. Really good.

Jean-Francois: I will have to pay attention to the appearance of the sunshine at some point because my exposure is entirely manual.

>> And the last protecter today, Jean-Francois Fortin Tam. Welcome back with bringing your favorite GNOME desktop application back from the dead. Please.

Jean-Francois: Well, thank you for having me. Pleased to be here. So, let's start. This talk is mostly a case study and a, you know, some sort of general guidelines on how to rebuild and manage your favorite free and open source project community. Or how to structure a project to avoid overload and burnout. Looking good so far? No crash? No -- seems to be good. Nobody complaining. All right.

So, most of you are probably aware that I spent the last few months of my spare time and energy bringing the Getting Things GNOME back and project back to life. So, this month I'm pleased to say, mission accomplished. So to speak. Because the project now has a new release and there is a community forming around it again for the first time in 7 years. So, in this talk I will cover the guiding project management principles I have put together to achieve this kind of result.

But before we get into the general principles, let's take a look -- let's take one step back and look at the context of the GTG project. And to revive a project like this, you need to understand why it died in the first place. Because if you don't, you're doomed to repeat the same mistakes. Especially if you have this superhero attitude and in the cursed words of engineering, "How hard could it be? Really?" .

So, this is a story of death and rebirth. Don't worry, it's not as complicated as a certain Japanese animation series from the '90s. GTG started 12 years ago. Like many free and open source software projects as a personal tool for two Belgian developers who probably had too much beer. It was an extremely well of had designed application and it became hugely popular.

And it was popular not just because it was a piece of technology defined before for the betterment of mankind. Which in the process of doing so, made you inspect the deepest recesses of your mind and re-evaluate your priorities in life. It was also popular because it solved a real need that people had in their everyday life.

So, GTG was a beautiful, revolutionary piece of technology at the time and it worked extremely well for its users. I'm living proof of that because I have been using it for like 10 to 12 years. And so, in the early years, many people contributed to the project. Since it was highly popular.

So, why did the project stall? What happened to the GTG team? Well, you know, after the initial development phase, the project became stable. And a set of features kept growing and growing. And all of that was going fairly well. Until the point where GTG had to undergo a big technological change which most tested Python applications can relate to, as the perfect storm to throw you into Dev hell. The infamous migration to GTG plus introspection plus Python 3 at the same time.

And so, like many other projects, GTG made the mistake of not only trying to port GTG3 into Python 3, but to fix everything else as well. So, two years into that at some point the maintainers burnt out and the project sat there unfinished.

And that's kind of the most tragic part, really. Is they were over 95% done. They were like this close to the finish line. But they didn't realize that back then. They had no clarity of scope. And they were in the mindset of trying to reach perfection. And, well, perfection is the enemy of greatness.

And meanwhile, it can very much imagine what was going on in the mind of potential contributors looking from the outside. It was obviously too scary to contribute because of a bunch of reasons. One, it was not clear what needed to be done to truly help the project. And secondly, you had this chicken and egg situation where people would look at the project in Dev hell and think, oh, boy, I'm not getting to -- I'm not falling into that trap. I'm going to wait this one out. It looked like this picture like this. And so, on the other hand, when the project was maintained, they probably thought, well, you know, someone else is taking care of it already. So, and when it became unmaintained, they thought, well, why would I succeed where others have burned out and failed before me?

It's a bunch of catch-22s out there. So, what do you want -- and what do you do when a project spent years trying to please everybody and to attract contributors and it doesn't work? Well, how you break that vicious cycle and ensure that people actually do get involved instead of staying on the sidelines... to do that, you need a very different approach to managing the project.

I had come to a point where the project needed a turn around. And a turn around is about doing a drastic shift in direction in order to save a business -- let me just adjust my exposure. Yeah. That's slightly better. Yeah. In order to turn around a business or save a project. And the beauty with the turn around is that there are no sacred cows. Either you turn the project around or you get a release out by sacrificing some features. Or you stay stuck in the same problem where you never make a release. But you have nothing to lose, really.

So, how do you do it? I mean, should you just ask the public for money to maintain the application? That's one of the most interesting ways, if you can pull it off. But the reality is that very few projects, particularly on the desktop, have a user base willing to pay for development and maintainership. And this is why I did a survey to evaluate the market demand for crowdfunding GTG development. Which quickly told me that was not going to be a viable approach in this case.

So, the alternative to the feature-based pay the maintainers approach is to do more with less. And adopt what I call a lean and mean agile approach by doing two things. One, reduce the software inventory. You can see my GUADEC 2013 talk. And the blog post that's mentioned at the bottom of that slide here. And the second thing is, you should start relying on the community much, much more for improvements. And any other critical bugs. Anything other than critical bugs, actually. And you need to provide the tools and the motivation for that community to get involved.

So, you need a product. And it needs to almost work. You know? In 2011, Havoc Pennington wrote an essay called, it has to work. And by work, he meant the intended customers really use it for its intended purpose. And GTG indeed does work. It has been working for me for a decade. But when you're doing community management in the participatory world of free and open source software, there is a twist to the it needs to work. As Linus Torvalds once said, don't expect people to jump in and help you. That's not how these things work. You need to get something halfway useful and someone will say, hey, that almost works for me and then they'll get involved in your project.

So, it has to work. Because if you don't have an attractive, mostly-working product, nobody's going to contribute to it because it doesn't already solve the majority of their problems. And this is one of the many reasons why I strongly believe that starting a new project instead of contributing to an existing one is a foolish move 99% of the time. And that's also one of the reasons, therefore, why I tried resurrecting this existing project.

So, if you've ever written a long-term roadmap for a community-driven open source projects, you will most likely understand the point I'm about to make. And my point is your hard-coded roadmap is BS. Any static roadmap is wishful thinking at best. It's a pure fabrication that is usually a waste of time to create and maintain. Your bug tracker's milestones is your true roadmap. And there you can simply and only target the things to a milestones when they are either already fixed or when you have a someone who is actively working on a fix or has assigned themselves to an issue with a believable promise of solving the issue soon.

Never target issues to milestones. Otherwise, leave them fluid or up or the taking. Don't have more than two milestones. Because nobody sees that far into the future. And use labels and priorities in a very specific way. We'll get back to that later in this presentation. If you look at the GTG roadmap page, I will candidly say it's the most honest software project roadmap seen. It doesn't commit to features and time frames because it realizes that you are wholly dependent on independent contributors to make that happen.

And now, another point I want to make is that for many projects, time-based or feature-based releases don't work. Particularly I want to say, screw the notion of time-based releases. I know this is going to sound really blasphemous as someone who is part of the GNOME community. But I really feel like time-based releases are -- for some type of application projects -- more harmful than useful.

Because, basically, they are arbitrary, they're constraining and they never match your own development canons. They mull you into the box of trying to land features based on a cycle. And to stash features that if you can't finish them in time. And more particularly, they can make your development cycle longer because you're releasing when it's time instead of releasing when the basic works.

So, you limit yourself to two releases a year, for example, where as you could theoretically release each quarter, each month, each week if you wanted to. If you're able to atomically deliver something. So, in my humble opinion, the MVP, or minimum viable product approach, is the way to go. And I also want to say, screw the notion of stable versus unstable releases. You either give your customer or users something that works or you don't. I mean, to do that, your desktop application is supposed to work and to be shippable at all times. Barring new API-dependent features like a new version of a library, nothing should stop you from releasing where you feel like it. And you should never break master. You should never break the master branch because that's what work in progress branches are for on private individual repositories.

And not breaking a master branch means you and the others can run the Git version for your own daily use. And it turns -- that means you can release atomic working tested products more quickly and more reliably.

Cause, you know, what is a release anyway? When you always have a continuously working and shippable development version, there's no such thing as a stable or unstable release, really. Okay. So, is there anyone here who is old enough and nerdy enough to know what this thing in the background is? I'll give you a hint. You can use a tent with it. See if anyone can guess this. Going once... going twice... okay. It's a safe point from Final Fantasy VII. All right. So, you know, Sigmund Freud once said that sometimes a cigar is just a cigar. This is a safe point. It's get tag on the comment and call it release-worthy. A release should not be the expensive thing to do. The most time sensitive was release was the writing and publishing of release notes and the announcement and trying to spread it like wild fire through the press and social media. I've done that only because it was a ground breaking release that had been expected for years. That's my point. Stable versus unstable. Maintaining more than one release at a time. Unless you have enterprise clients, it makes no sense.

If you are a community-run project working on the desktop application, free your mind from the arbitrary roadmaps, the assume schedules, the release denominations. These are all artificial constructs and they are the opposite of agility.

You don't need scrum, really. I think scrum is a buzzword. What you really need here is to be decisive and clear about the scope of what you want to do. And what is a minimum viable product that you can release? And that's how you get to be lean, flexible and agile.

So, when you free your mind about trying to fix everything, you can do this. This is loosely -- I mean, I have a philosophy. Loosely derived from Joel Spolsky's philosophy, fix the corruptions, the grossly broken main features. Until you feel like you're fixing silly bugs. You don't fix the silly bugs, you make a release instead. Either severe bugs will come back and -- or you will get new contributors to fix the silly bugs. But at least you released and you fixed the critical bugs.

So, the -- the MVP eternal "It almost works" product development myth requires clear expectations and therefore, documentation. So, you have to make it clear that you are not fixing the silly bugs or feature requests yourself unless you're feeling particularly bored and feel like doing so. And rather, you wanted to make it clear that you are relying on new contributors to make that happen.

So, in the bug tracker, you use priorities and labels intelligently with a very clear intent and process. It is critical for crashers -- you use basically the critical label for crashers or data corruption. And higher priority -- and high priority is for very broken existing major features. And low priority is for silly bugs.

Medium is for basically anything else. It's for the regular bugs that have a work around or that are not core broken features. And then you use the low-hanging fruit tag, or the patches or won't happen tag. And using those, you tag everything that you can't promise to fix yourself. Especially things that you can't fix yourself in the near future. You leave them up for the others. And then you as a maintainer or a core developer, when you look at the fix of bugs, you filter down to only see the critical bugs. And you are very selective about what is allowed to get labeled a critical bug. Or a critical, or a bug.

If it's minor or a normal bug or a feature request, shouldn't be in your view unless you're already made a release recently and you have run out of important bugs. So, you prioritize the bugs that impair the ability of contributors to do their work efficiently or things that really break the application.

And, of course, as you can see on the screen, you have to really explain the meaning of all these things in your contributor's documentation. Again, it's all about managing expectations. And for this kind of revival thing to work, you need at least two players. Okay?

Cause, sure, over-specialization kills. But specialization helps as well. You know, most humans are not built for multi-tasking. How you solve that? Well, it's a hardware problem. The solution is adding more hardware in this case. And that means more humans. So, you need an accomplice or two. So, that you can each specialize in some areas if you have different skill-sets and run at 100% each. You don't need 30 people. You need at least two.

In my case, I was specializing as a marketer and product manager. And Diego was specializing in fixing bugs. I helped with bugs a little bit. But it was not my main motivation. As a project maintainer, you want to be prepared for a lot of yak shaving.

Yak shaving is any apparently useless activity which, by allowing you to overcome intermediate difficulties, allows you to solve a larger problem. Daniel Stone, Wayland developer, is a notable example of a yak shaver. I actually have some yak fur here to prove that I've done the same thing with GTG.

So, yeah. Some more recommendations, globally speaking. You want to focus on removing the impediments to contributing. So, focus on the contributor experience which is going to mean a lot of bug, documentation, packaging, infrastructure planning. And all that yak shaving needs to be towards the goal of making the project sustainable so that it can continue to attract contributors and be autonomous in the long run. You need to do stuff that will multiply the productivity of other people.

And second recommendation I have is market and communicate what you're doing. So that people can buy in and early adopters can hopefully join the project. I was extremely lucky to have had Diego join the revival project early on. It was highly motivating. And after I announced the results of the survey. So, it was a month ago. But he would probably not have joined if I had not communicated the work I had started to make this happen.

Another recommendation is to focus on the singular goal of getting the release out with the new contributors experience. Anything that detracts from that goal such as, you know, a new logo, a new name, fixing plugins, the sync backends, making a new website. Skip all that. Because you don't want to waste time -- Whatever little time and energy you may have yourself or from potential contributors on things that don't get you closer to that goal.

The clock is ticking for you as a maintainer. So, you have to work with a sense of great personal urgency to create and maintain momentum. You have to stop starting things. You have to start finishing them. Multitasking it for CPUs, not humans. And you can find some more details at the bottom of the slide where there's a link to a blog post of mine.

And, you know, I think it was Christian, maybe? Who many years ago once said to me something like, where are all the good project managers in free software? We need them. Because otherwise as developers we sometimes tend to go into ten different directions and get overloaded. And it's much easier to focus on coding if someone takes care of the rest.

The sun is coming out. And I need to adjust my exposure. There we go. All right. And then the thing with that is I would argue that, you know, many competent and humane project managers are not involved hands on in community-driven free and open source software because of the personal cost. It requires a tremendous amount of dedicated work. Nobody pays them. And in a code-centric culture, their contributions are rarely seen nor recognized. And you kind of need the right set of conditions.

You know, essentially, a crisis situation for a team of developers to fully understand the value that they add. And to entrust the leadership to them and follow their guidance. And truth be told, not everybody is equipped or prepared to do that kind of grassroots work working under those circumstances. So, in practice, market forces typically make it quite unattractive for those type of people to get involved.

With that said, I think I have like 60 seconds left. So, I would consider that my talk is mostly over.

>> This is the last talk, you can talk a bit longer if you want to.

Jean-Francois: Oh, so, I can continue for like four hours.

>> We have social hour in 14 minutes. But yeah. You still also have a lot of questions, actually. So --

Jean-Francois: Wow. I think this chat has been going on for quite some time and I was looking from the corner of my eye thinking if I would time to answer this. I'll try to. And I will also say while I'm here, if you or a colleague is looking for a great marketer or product manager or some other kind of person with a deep understanding of free and open source software. I'm available for hiring and contracting. So, anyway, you can reach out to me. There's the link to my website at the bottom of the slide. Let's see what questions or insults people might have for me in the chat. Let me scroll back up.

>> You can find the questions actually in the shared notes. If you click on shared notes and click on Etherpad link at the bottom.

Jean-Francois: Right. Let me lag a little bit in the web interface.

Do, do, do ¶

Closed captioning -- no, Etherpad. Okay. Oh, okay. I still have to scroll those notes down. Let's see. So many colors. So little time. What's your most anticipated anime for year 2020? I would say rebuild of Evangelion. I'm not sure if it's going to happen. I'm proud to have beaten rebuild of Evangelion with the rebuild of Getting Things GNOME, basically. I haven't watched anime in years, actually. I blame my never ending to do list. Will you be looking forward to the guiding newcomers and students for GTG? I'm always welcoming newcomers to the project.

And in fact, after the release in the last week or two, I have been seeing some very encouraging interests not only from users, but at least two or three people showed up saying, oh, I would like to work on this plugin or work on a new plugin. Some people are attempting to work on new online synchronization features. No promises yet. But it's looking interesting. We'll see what happens. We can always welcome more help.

Students, if you mean summer of code-type of students, I haven't registered to gStyle Core or anything like that. So, clearly not this year, but, hey, it would be next year or something. We'll have to see.

Have I ever regretted taking on the resurrection of Getting Things GNOME due to the mess it was? No. Because that's something I really wanted to do. Sure, sometimes I thought, gosh. This is taking a long time. But at the same time, it's software I love. I have been using it for 12 years, you know? I built my life around this software. I can't let it die. And actually, that's -- that software is the reason why I was talking in a previous release of Fedora for a while until a Flatpak was available. A very secret Flatpak. Thank you, Bilal. And, you know, I thought -- I took it as a personal challenge in a way. So, it was also fun to do it. Per se.

And seeing new contributors show up, especially Diego in the beginning, that was really encouraging. Because when you're not alone in this thing, it's -- it drives you forward. And seeing the reaction online when the release came out and seeing new contributors coming up, that's -- that basically makes it all worth it.

And, hey, I get an application that works in the process. And it works better than it used to. And has a nice UI. So on. Mission accomplished. I don't have to -- to use paper or something. I don't use paper. Paper sucks.

What was the hardest part of reviving Getting Things GNOME? What could we take as a learning for doing the same for other applications that we want to refresh? The hardest part -- I don't know. I guess you have to be -- you have to know what you're doing kind of. You have to be realistic about your expectations and you have to kind of tests the waters to see if people are going to be interested in helping you with this. Because if you're going to do the same thing that others tried to do before, as I said. If you try to do everything. And if you try to do it alone, you're not gonna get very far. You're gonna burn out, probably. Because if the project went unmaintained, probably it was because it was in Dev hell. And that's what happens to most of the projects that go unmaintained. So, you kind of have to go there with a level head. And to know what are your personal limits. And, you know, have realistic expectations. In my case, I thought the only way to make this work is to try to get a minimum viable product out of the door and try to resurrect the community, focus on the community.

And then we can work on all the fancy features eventually. But if you don't get the minimum viable product out, tough luck. I also knew that while I know Python, and I could technically do all the coding work myself, I wanted to structure the project in a way that it's not depending on me coding everything. Sure, I can throw a patch here and there sometimes. But that's not where personally I add the most value. And I kind of lost my train of thought here.

But, yeah. You have to be... you have to structure it in a way that you know the project is not gonna be resting all on your shoulders. Because that's going to be an immense weight to carry if you think, well, who else is going to keep the project alive if I get hit by a bus? Bus factor of 1. That's my horrible phone ringing tone from the '80s. Okay.

>> Okay. So, that's -- that's all we have time for the questions. Leave the rest of the questions to the chat. Thank you, Jean, for the really nice, really beautiful presentation. This is the last presentation for today. And we will continue in less than -- more than 40 minutes with the social event, cooking together with Sriram Ramkrishna. Welcome back in 40 minutes. And thank you.

Jean-Francois: Thank you all for your attention and your questions. Sorry I couldn't answer the last few ones. So, have a great end of the day. And looking forward to see Sri cooks up.

>> Thank you.