Open Source Development and Sustainability

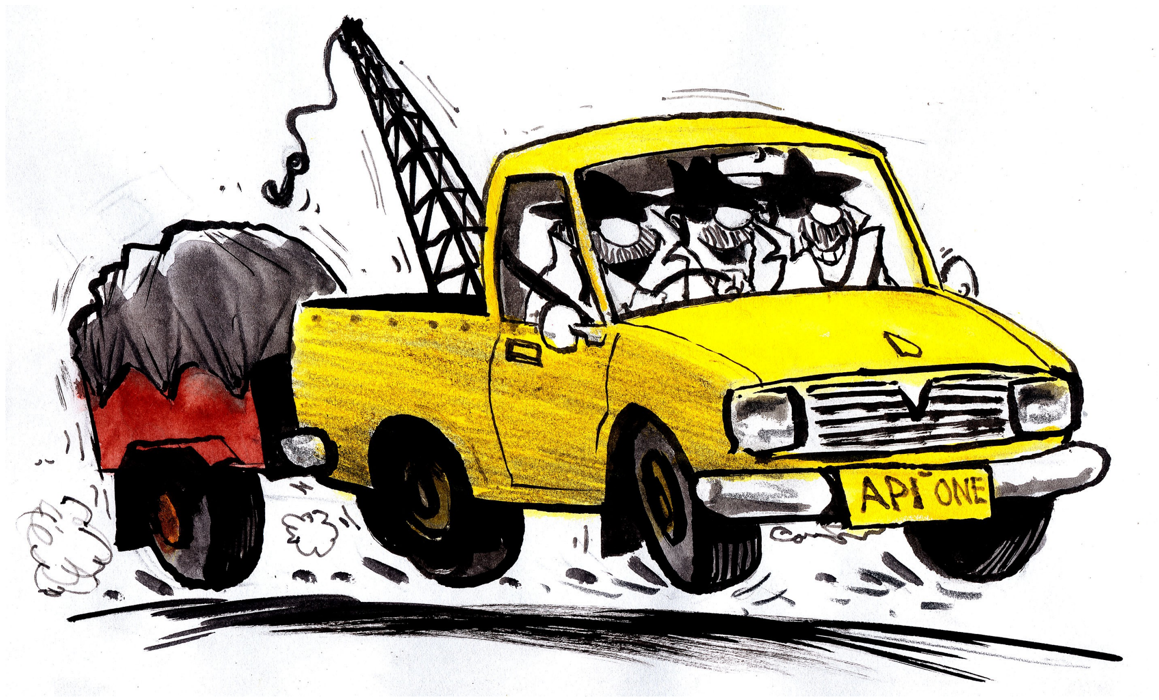
A Look at the Bouncy Castle Project

How It Started

Early Days

- Started with a low level API as one of us was playing around with the J2ME, built a provider on top of it.
- Added functionality for generating X.509 certificates.
- Then, of course, CRLs.
- Over the next couple of years, a few more algorithms (e.g. Elliptic Curve), improvements and additions (PKCS#12 support), and then...

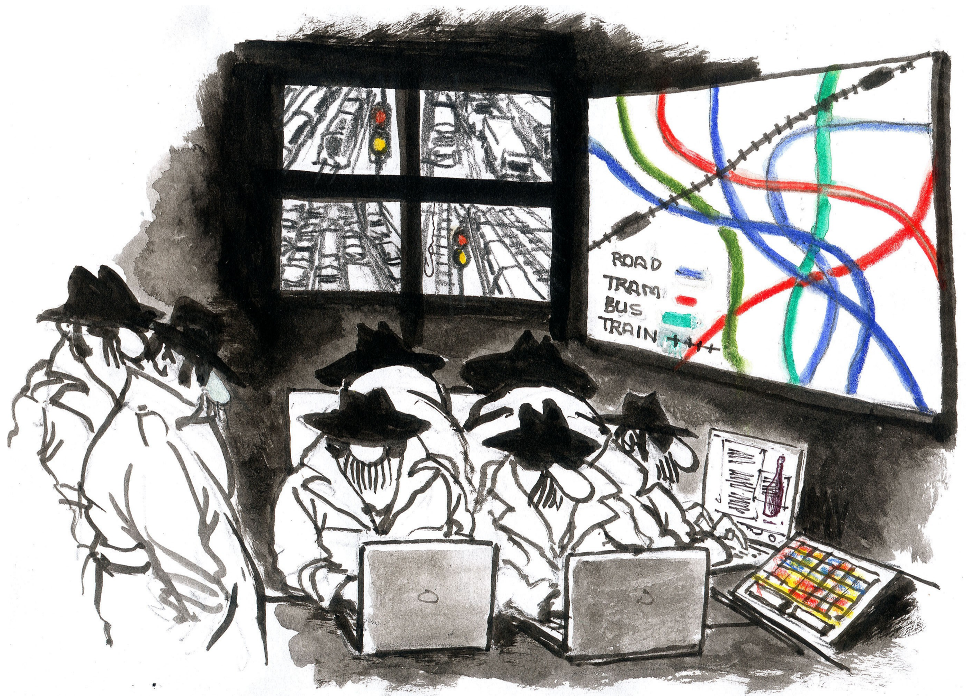
Really Up and Running



More Features!

- Support for Cryptographic Message Syntax
- Support for Time Stamp Protocol
- Support for OpenPGP
- Attribute certificates
- A broader range of standards (paddings, algorithms)
- And more! But...

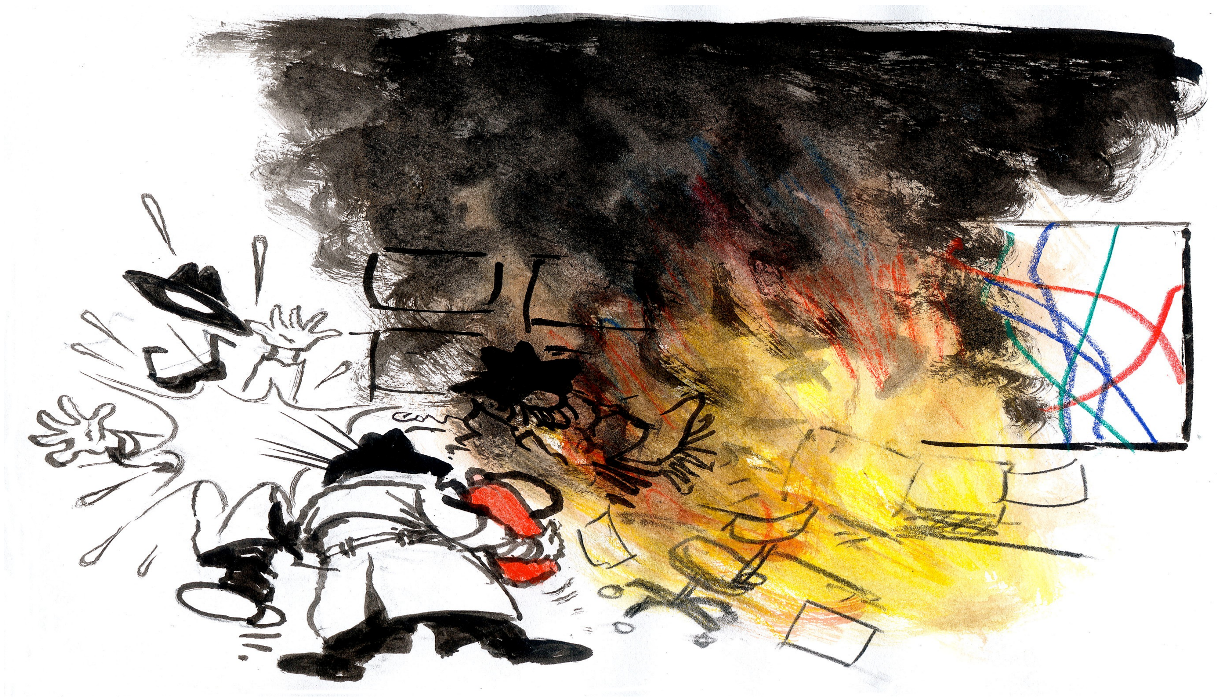
Suddenly, There is Complexity



And Realisation

- It's no longer just something you put on the Internet because you found it useful and thought someone else might.
- People are actually relying on it.
- You even find out your bank is relying on it!
- Reviewing the situation in this light, one rapidly realises that as good as everything is, it's one step from...

Chaos and Disaster



Principal Constraint - Time

- The issue isn't really ever money – it's actually time.
- Money does help free time up but is not a solution in itself.
- Lack of time can result in poor, or incomplete, test coverage, hasty check-ins, incomplete functionality.
- Favourite brother to “lack of time” is interruption. Interruption on second tier work is often and generally lengthy.
- Often open-source work has to be treated as second tier.

Other Constraints

- Equipment – faster computers, quicker turn around, servers for continuous testing.
- Infrastructure – issues trackers, mailing lists, website, managing distributions, download areas, 3rd party deployment (e.g. Maven central).
- These days, the costs for most of these are modest, again the issue is time, time to administer and time to make use of what infrastructure you have.

Other Problems

- Ideally people outside the core team should be able to contribute.
- Suddenly it takes as long, sometimes longer, to review a contribution than it would to do it locally.
- Large contributions become especially problematic, particularly if they involve standards such as ASN.1, as even experienced developers frequently make errors.

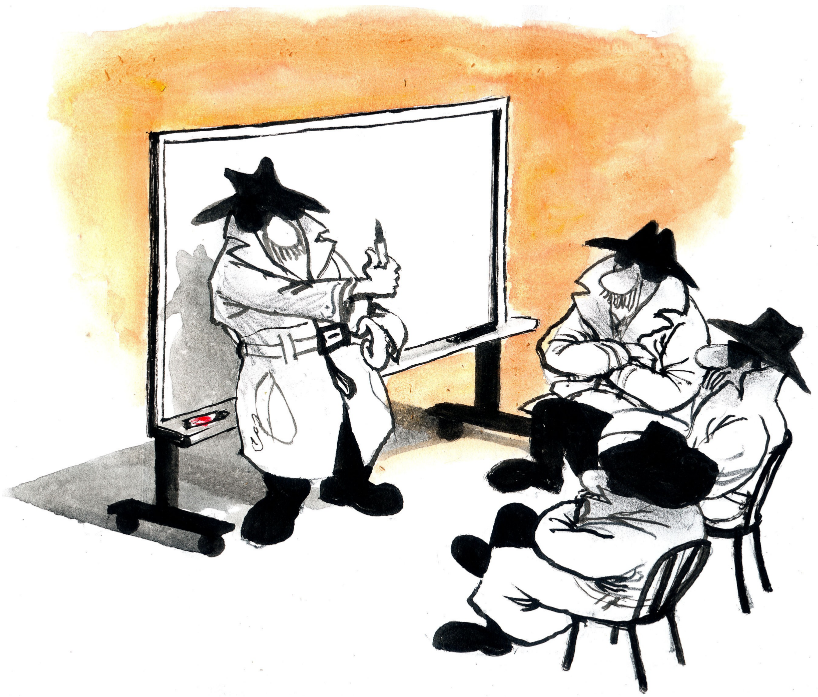
Biggest Danger

- Accrual of technical debt.
- A big issue in security orientated software as in some cases things can go from being great to useless, possibly even dangerous, overnight.
- Can also result in code which is difficult to maintain, again making it difficult to respond to changes.

Then of Course...

- Peoples' lives can change
- External pressures can change
- Children, dogs, cats...
- The bank that's using your software suddenly becomes the one that also holds your mortgage.
- Is this really what I signed up for?

So What to do?



Immediate Thoughts

- Rely on donations?
- Maybe a product company?
- Fund through consultancy work?
- Change license?
- Public/Professional version?
- Run?
- Before doing any of the above, need to consider what you want to preserve as well.

In our Case

- Decided not to run.
- Wanted to preserve open source. Openness the best approach for cryptography software.
- Donations unreliable. Not tax deductible in themselves.
- License fees, community/professional model not really an option. Can't do “partial” cryptography, risk of introducing errors unacceptable.
- Contracting helps a bit, but have to be careful as it rarely means working directly on the APIs. Doesn't buy much time.
- Product built on API approach also problematic, same issue as contracting.

The Solution

- Established a charity with ownership of the code base.
- Established a company for actual commercial work.
- Really had to find a way to make the APIs and the “product” related.
- Only accepted short-term consulting targeted to the APIs.
- Started selling support contracts.

The Product

- Turned out to be support contracts.
- Question then is why would someone buy a support contract?
- Some people will buy one because they want to support the project, or they actually know they need support.
- Most people need something tangible that's different from the public offering.
- In our case, early access to certification work.

Things You Wrestle With

- “Freeloading” - is that what's really happening and what does it mean?
- Do people really understand where the money goes when they buy software?
- Turns out “not paying” and “freeloading” aren't always the same thing.
- That said, there are advantages in having a large user base for a Crypto library if you can keep up with the users.
- These advantages also benefit paying customers.

Other Things That Change

- If something needs to get done, it cannot be treated as second tier work.
- Different risks emerge, a lot of knowledge in the heads of too few people.
- To deal with these it means the project needs to expand, and people need to be paid.
- Not only have to manage the code, but manage the knowledge.

It's not just the code base we need to preserve!

On Reflection

- Many of the issues are really the same you face with any business.
- If you need an income, you have to have something to trade for cash.
- In commerce everything is quite simple, but even simple things can seem quite difficult...
- If you are running, or setting up, an Open Source project you should think about these things early.

Thanks for listening.

Any questions?

Open Source Development and Sustainability

A Look at the Bouncy Castle Project

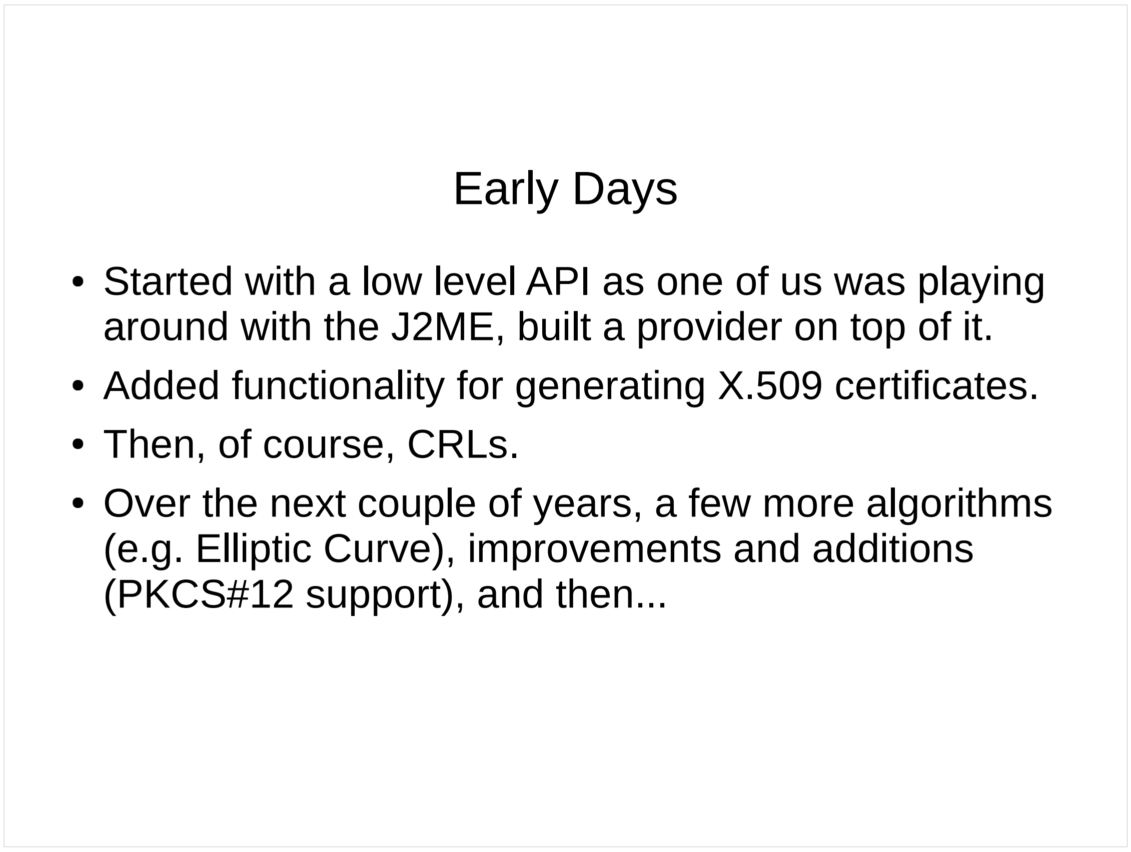
How It Started

Project started as a result of a previous effort which got turned into “abandonware”. At the time SSL restrictions were being lifted but while a standard API for cryptography in Java, the JCE, had been announce, Sun, the original Java vendor, were unable to export it from the US. Australia on the other hand, had no restrictions in place on the export of crypto code on the Internet.

Project is informal, writing stuff up with test vectors, trying things out to see what plugs together.

At the start there isn't much planning the code base grows organically. Work gets done on weekends, over beers, at dinner.

No real pressure to get anything done, no weight of expectation.



Early Days

- Started with a low level API as one of us was playing around with the J2ME, built a provider on top of it.
- Added functionality for generating X.509 certificates.
- Then, of course, CRLs.
- Over the next couple of years, a few more algorithms (e.g. Elliptic Curve), improvements and additions (PKCS#12 support), and then...

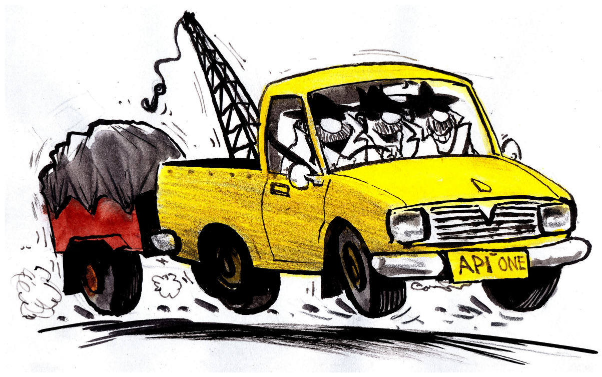
First attempt was actually due to the Palm Pilot. I think the actual app was about handling passwords on a mobile phone. A JCE style approach was tried but it was just too much. The “lightweight” API was born out of this.

The lightweight API then turned out to lend itself well to supporting a JCE.

Website went live in May 2000, 1.0 release was in October.

The ability to make certificates was a big selling point.

Really Up and Running

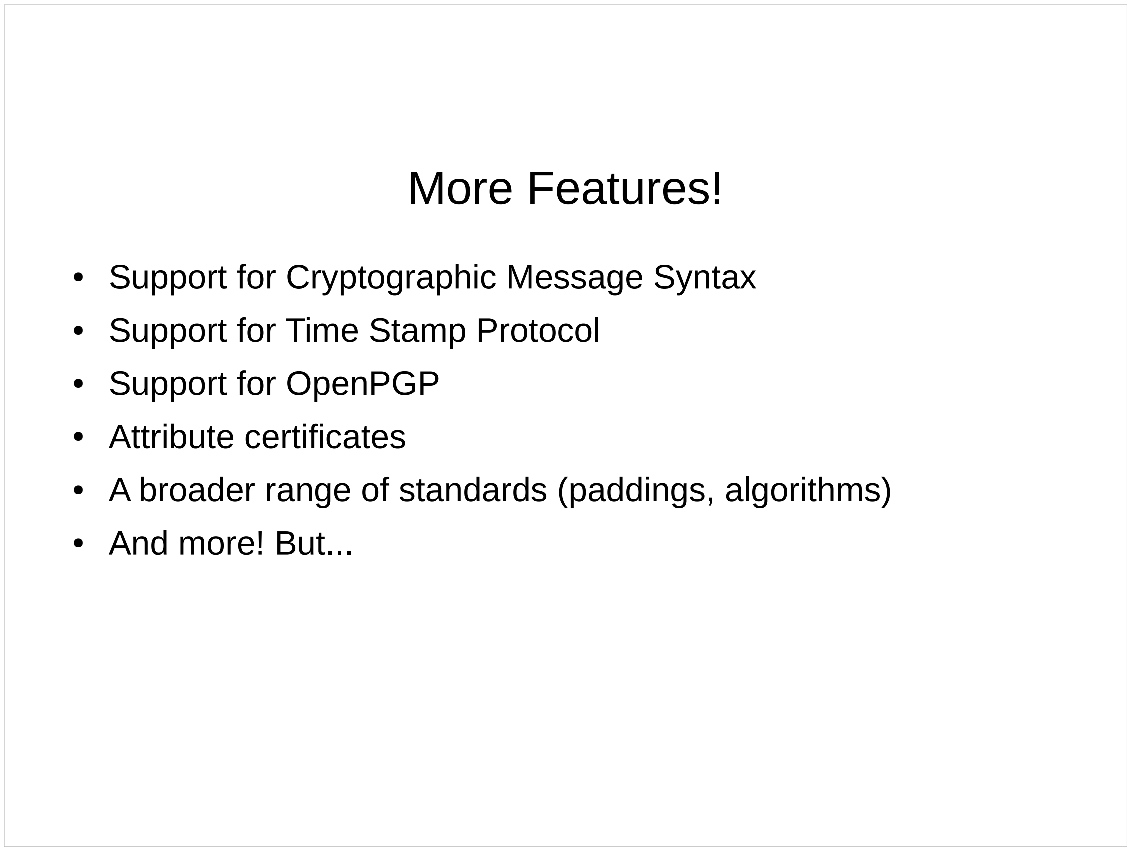


Sun eventually got permission to export by introducing the “jurisdictional policy files” and mandatory jar signing into the JCE.

The JCE was released as part of JDK 1.4 in Feb 2002.

The rules relating to open source cryptography had been relaxed by then and we were able to carry out the process required for getting an export license, and apply and receive a JCE signing certificate.

It was all systems go.



More Features!

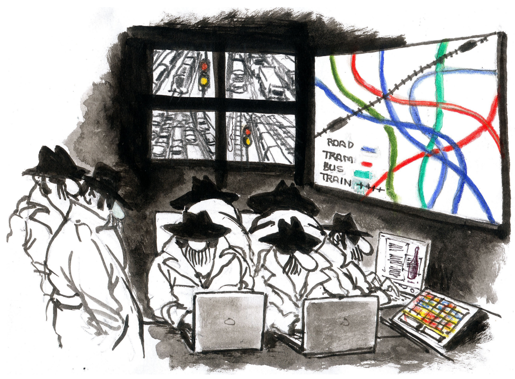
- Support for Cryptographic Message Syntax
- Support for Time Stamp Protocol
- Support for OpenPGP
- Attribute certificates
- A broader range of standards (paddings, algorithms)
- And more! But...

Originally we were very algorithm focused, as are most people when they start out. It became pretty obvious that the real struggle was about interpreting the various IETF and ANSI/ISO standards as APIs that people could use.

Supporting X.509 naturally led onto CMS, PKCS#12, and various others. Once we'd added a few things, people were a lot more interested in suggesting and occasionally helping with others.

An API for OpenPGP was added towards the end of 2003.

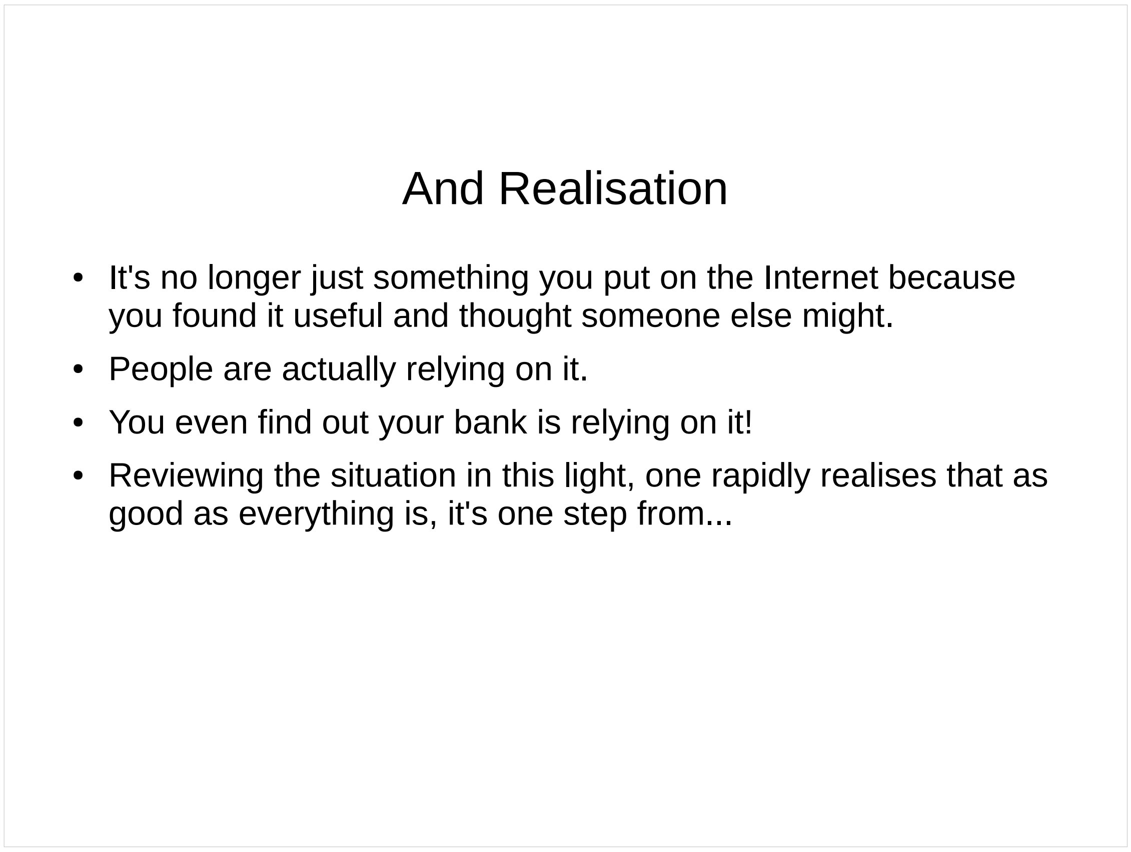
Suddenly, There is Complexity



APIs for different standards continued to be filled out and adjusted as the standards themselves change.

For things people used a lot we rapidly got to complete coverage. Test coverage became more and more difficult though, lack of published vectors for different protocols, coupled with actual implementations from “real vendors” which didn't quite agree as well, meant it was quite a struggle to keep things working.

In 2007 a C# version was contributed, the Java code base was over 200k lines, from the original 27k.



And Realisation

- It's no longer just something you put on the Internet because you found it useful and thought someone else might.
- People are actually relying on it.
- You even find out your bank is relying on it!
- Reviewing the situation in this light, one rapidly realises that as good as everything is, it's one step from...

Okay, it's a cryptography API, so you'd think we would have seen this coming. Guilty as charged, that said it really was one of those “it's not quite what I imagined it would be” kind of things.

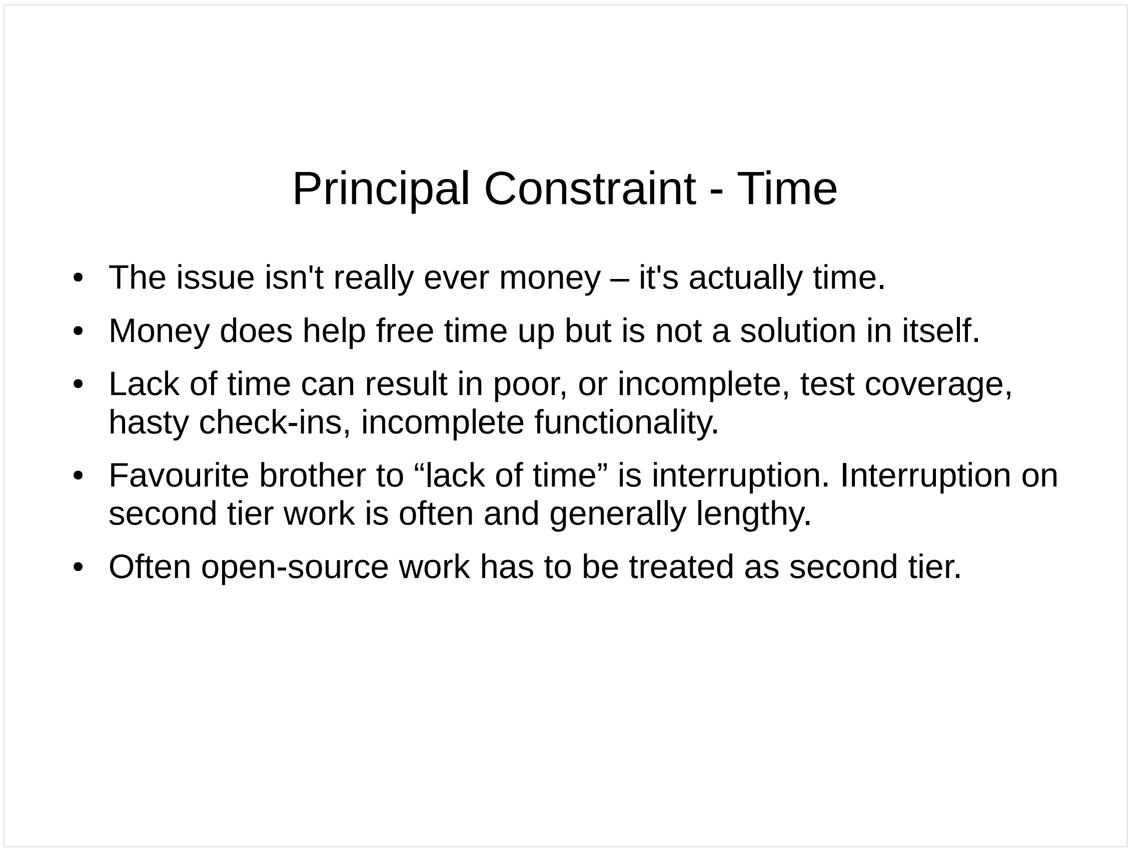
The APIs continued to expand, work issues made life difficult, family situation started to change, it started to get very difficult to get a release out the door. We were trying to aim for one every 3 to 4 months. Suddenly it was hard to get out one per year.

Chaos and Disaster

Actually I think the picture says everything, at any rate by 2012 it was clear we were not in a comfortable place at all. Release 1.46 came out in March 2012, we only just managed to get 1.47 out before the end of February 2013.

If something major had happened we would have been in a terrible position response wise.

By February 2013, the Java code base was 269,000 lines including tests. The C# API was 145,000 lines as well.

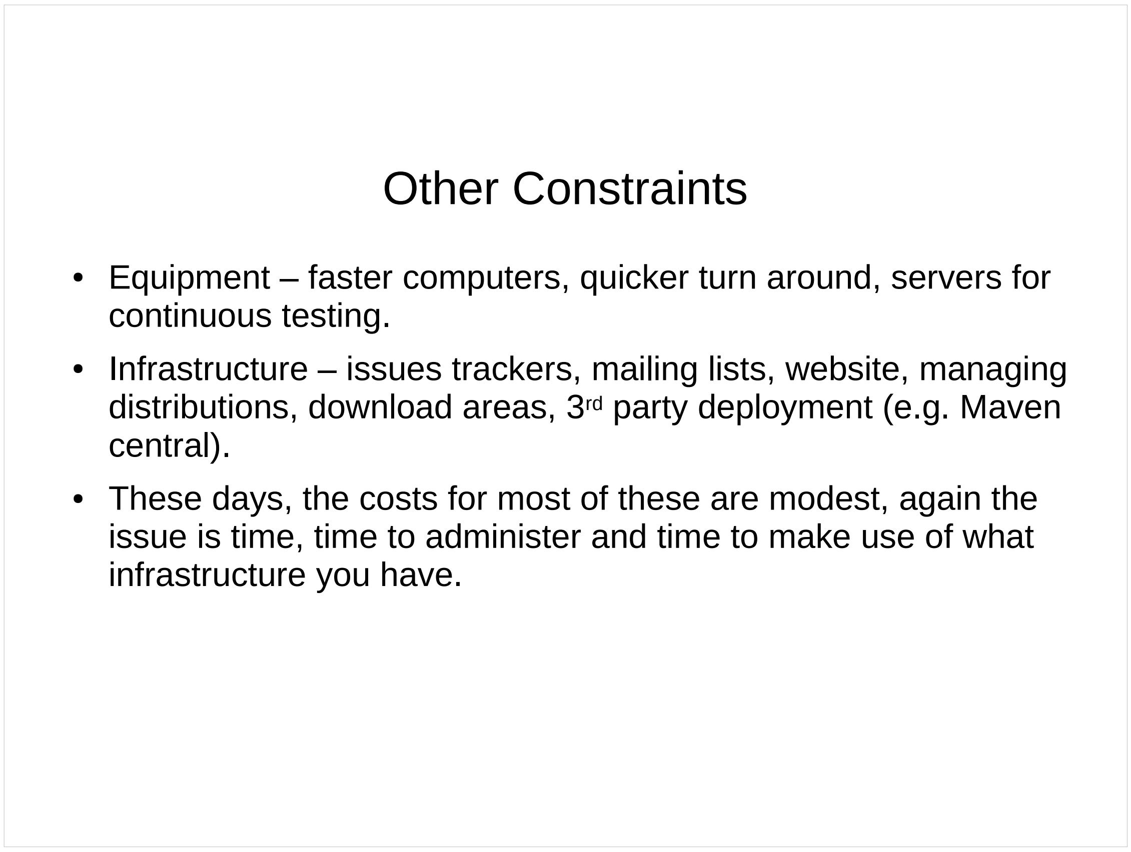


Principal Constraint - Time

- The issue isn't really ever money – it's actually time.
- Money does help free time up but is not a solution in itself.
- Lack of time can result in poor, or incomplete, test coverage, hasty check-ins, incomplete functionality.
- Favourite brother to “lack of time” is interruption. Interruption on second tier work is often and generally lengthy.
- Often open-source work has to be treated as second tier.

Programmers, especially reasonably accomplished ones are in demand. At that level we don't have a lot of trouble getting well paid (at least by some people's standards) work.

But if you've got a day job it's not normally just related to your “hobby” project, so a regular job takes up time, and individuals cannot scale.


Other Constraints

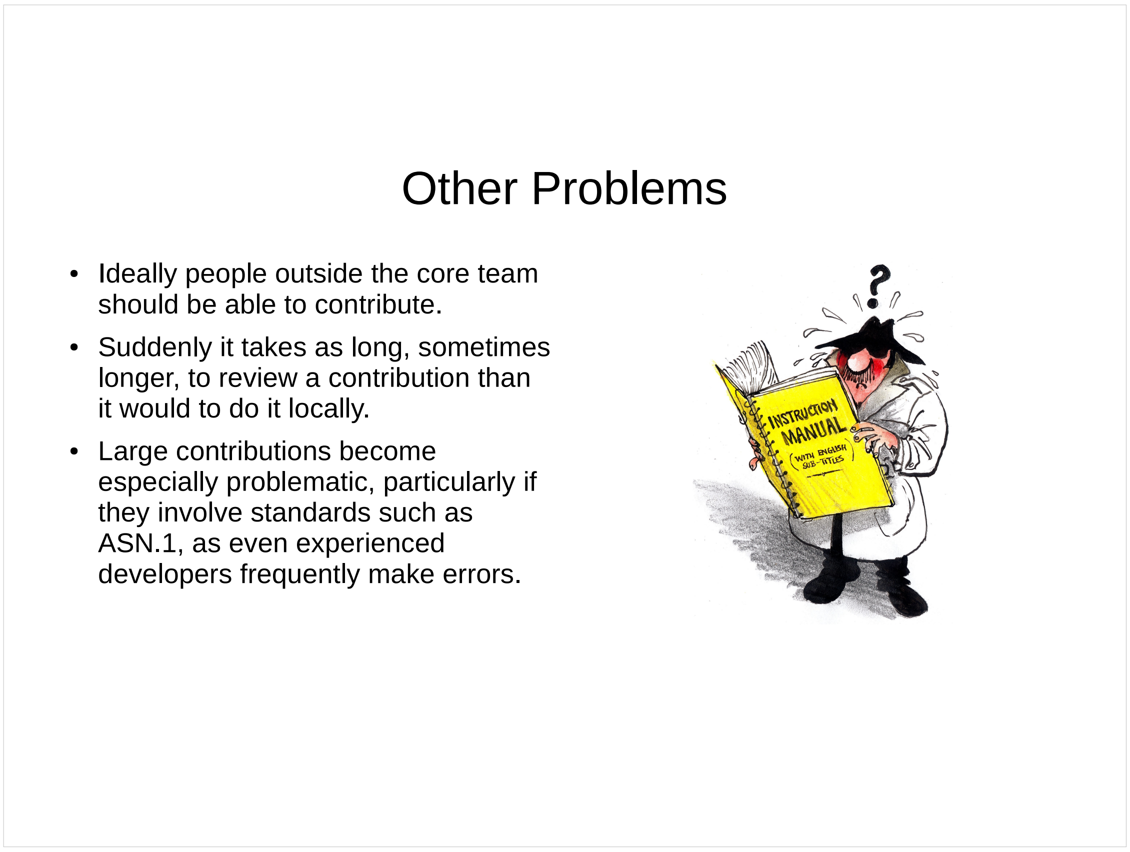
- Equipment – faster computers, quicker turn around, servers for continuous testing.
- Infrastructure – issues trackers, mailing lists, website, managing distributions, download areas, 3rd party deployment (e.g. Maven central).
- These days, the costs for most of these are modest, again the issue is time, time to administer and time to make use of what infrastructure you have.

In some ways this slide is superfluous, it could be replaced with the quote “Time! Time! Time!”, but it does provide some good examples of the kind of things that get you into strife when you are running an open source project.

Licenses for our issue tracker and wiki were donated by Atlassian, which is great. However, anything vaguely complex which is Internet facing requires constant maintenance. This is not so great.

None of the administrative task involved in infrastructure management are generally complex in themselves, together however they can represent a lot of time.

Source code management due to requirements for FIPS/Common Criteria do present some extra complexity as well.



Other Problems

- Ideally people outside the core team should be able to contribute.
- Suddenly it takes as long, sometimes longer, to review a contribution than it would to do it locally.
- Large contributions become especially problematic, particularly if they involve standards such as ASN.1, as even experienced developers frequently make errors.

As the project gets bigger it becomes harder to write a contribution, even developing one on a fork can be problematic as the trunk may drift.

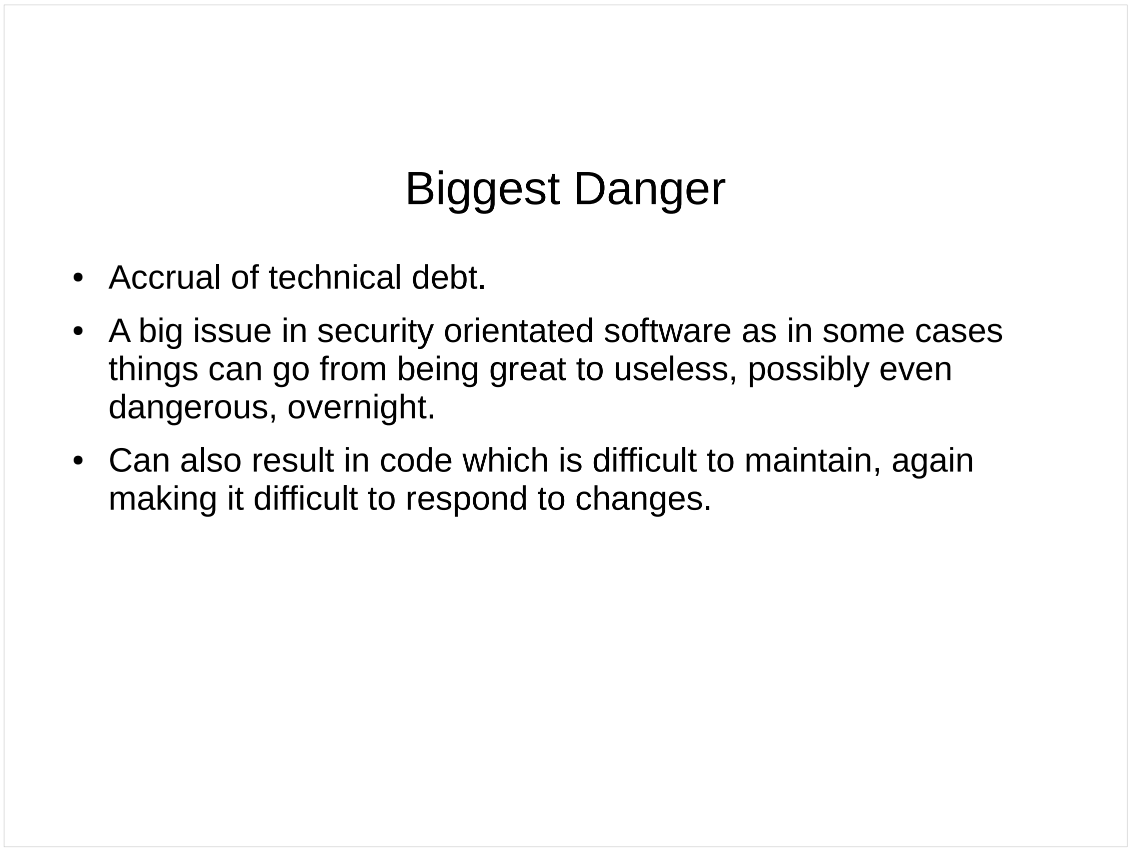
Submissions also need to be reviewed, this takes time, normally some parts of the submission get rewritten, this also takes time.

Often when people submit code they're solving “their problem” this is often not the same as the actual problem the standard the submission is from may be trying to solve, or what you'd do when designing an API.

Sometimes people will solve “their problem” and actually break something and not realise.

All these things have made accepting submissions progressively more time consuming.

Outsiders being able to submit code is a core part of the project though. It's a problem.



Biggest Danger

- Accrual of technical debt.
- A big issue in security orientated software as in some cases things can go from being great to useless, possibly even dangerous, overnight.
- Can also result in code which is difficult to maintain, again making it difficult to respond to changes.

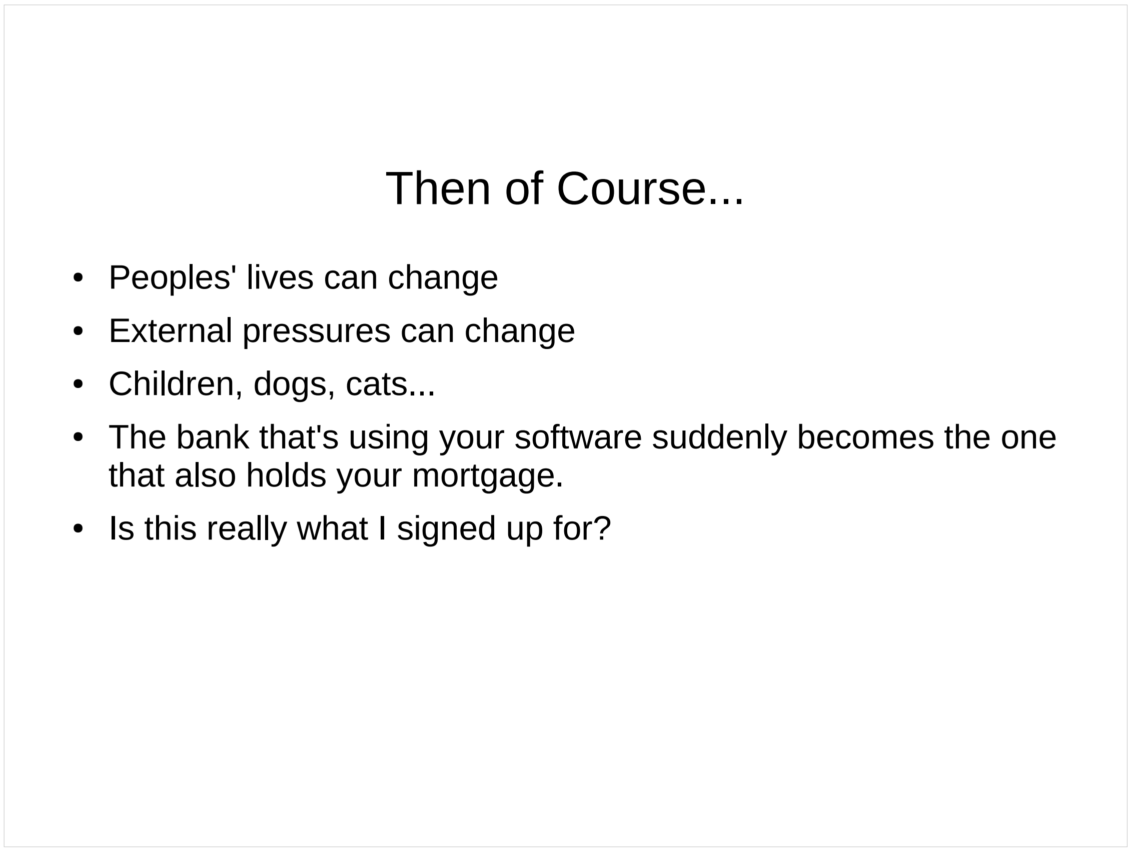
We've found that constant review is necessary. This also involves a certain amount of rewriting.

It is also important to be able to support new development, standards change, new algorithms and techniques appear.

Too much review/rewriting it appears the project is stagnating.

Too many additions and other enhancements and the code base gets progressively more fragile.

It's important, but difficult, to strike a good balance on this.



Then of Course...

- Peoples' lives can change
- External pressures can change
- Children, dogs, cats...
- The bank that's using your software suddenly becomes the one that also holds your mortgage.
- Is this really what I signed up for?

So when it started, everyone was “care free” and largely single.

Now several legionnaires have children, I think everyone's got a mortgage.

One of the issues with us “open source” types is that we generally embark on these projects because we think they're important at some level. So its normally a bit of a shock when an attempt to do some monetisation falls flat on its face – perhaps others don't really see it our way? So why are we doing it? On the other hand, maybe the project is important, but we're asking people to look at it the wrong way.

So What to do?

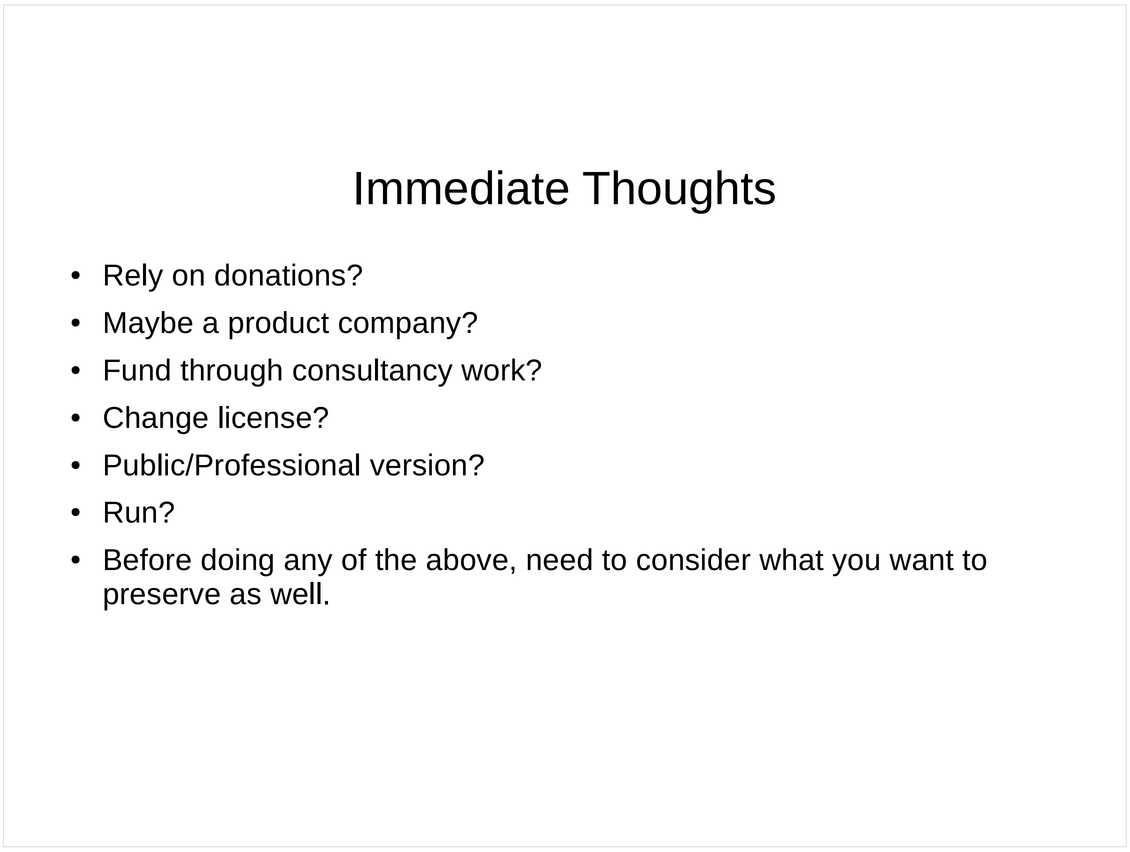
Well, at least appreciating you're in trouble is a first step.

The problem then is the second step.

It really does take work to start a business!

And it's not the same kind of work you're actually doing on the project.

To quote Jim Hightower, probably completely out of context: “Do something. If it doesn't work, do something else. No idea is too crazy.”



Immediate Thoughts

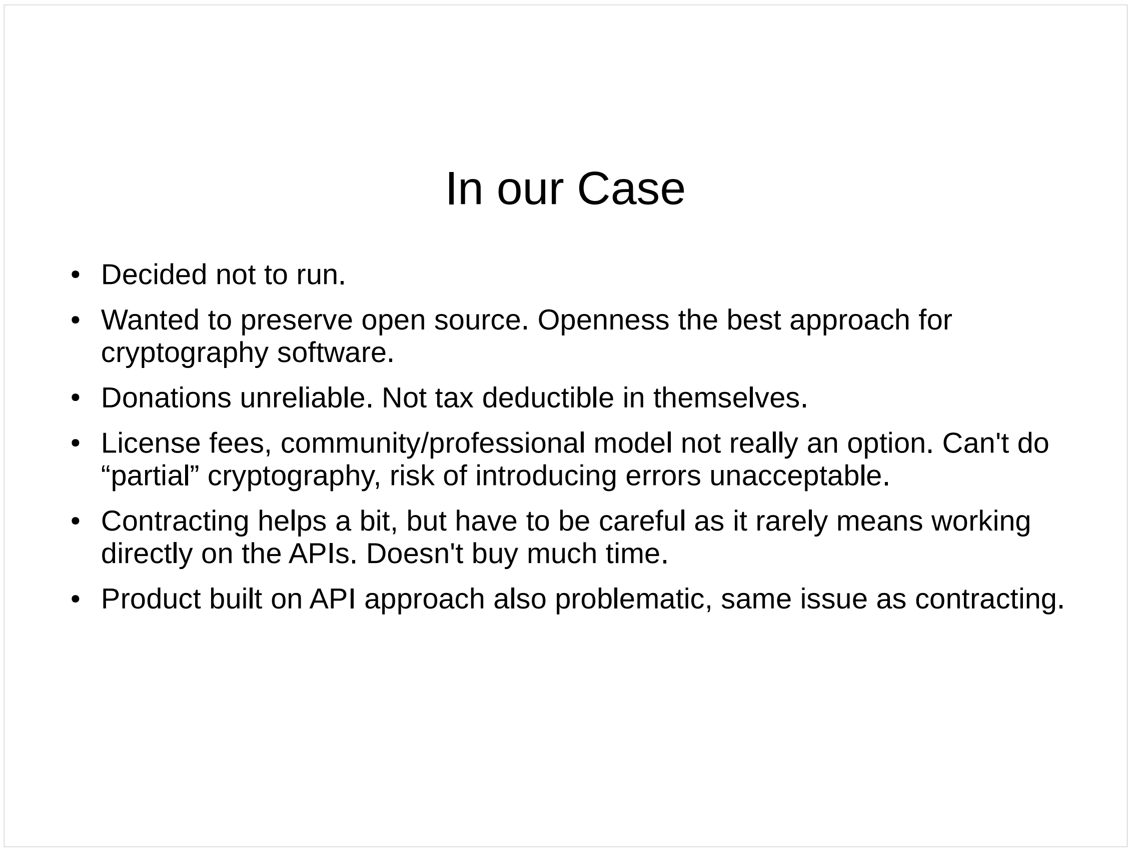
- Rely on donations?
- Maybe a product company?
- Fund through consultancy work?
- Change license?
- Public/Professional version?
- Run?
- Before doing any of the above, need to consider what you want to preserve as well.

Yes, running is always an option. Even SSLeay ended up as abandonware briefly before it turned into OpenSSL.

Other than running, everything on the list also involves how people outside the project think and feel about the project.

For that reason, as much as anything, there's no “silver bullet” answer. The best approach is likely to vary depending on what it is you actually end up selling to fund the work.

Kicking off a business is no small task though, you'd better be sure when you start you have some understanding of what you're trying to get out of it.

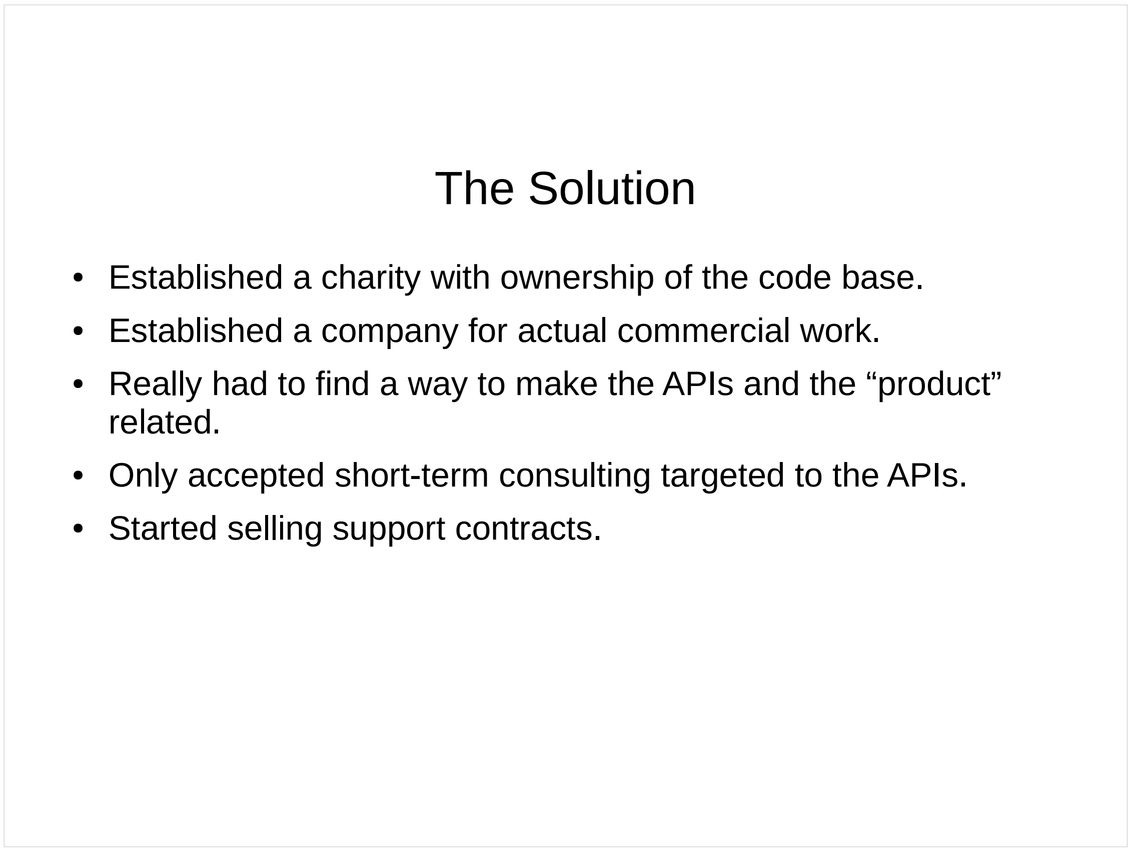


In our Case

- Decided not to run.
- Wanted to preserve open source. Openness the best approach for cryptography software.
- Donations unreliable. Not tax deductible in themselves.
- License fees, community/professional model not really an option. Can't do “partial” cryptography, risk of introducing errors unacceptable.
- Contracting helps a bit, but have to be careful as it rarely means working directly on the APIs. Doesn't buy much time.
- Product built on API approach also problematic, same issue as contracting.

One of the other things we wanted to preserve was to maintain people actually talking to developers when they needed help.

This is an interesting feature of Open Source projects, largely because none of us can afford help-desks! It does have a small down-side as it means a project developer does have to deal with “is the power plugged in” kind of questions, but there's a massive upside as users of the APIs can actually communicate accurately with project developers. Too many times client->help desk->developer turns into a game of “whisper down the lane”.



The Solution

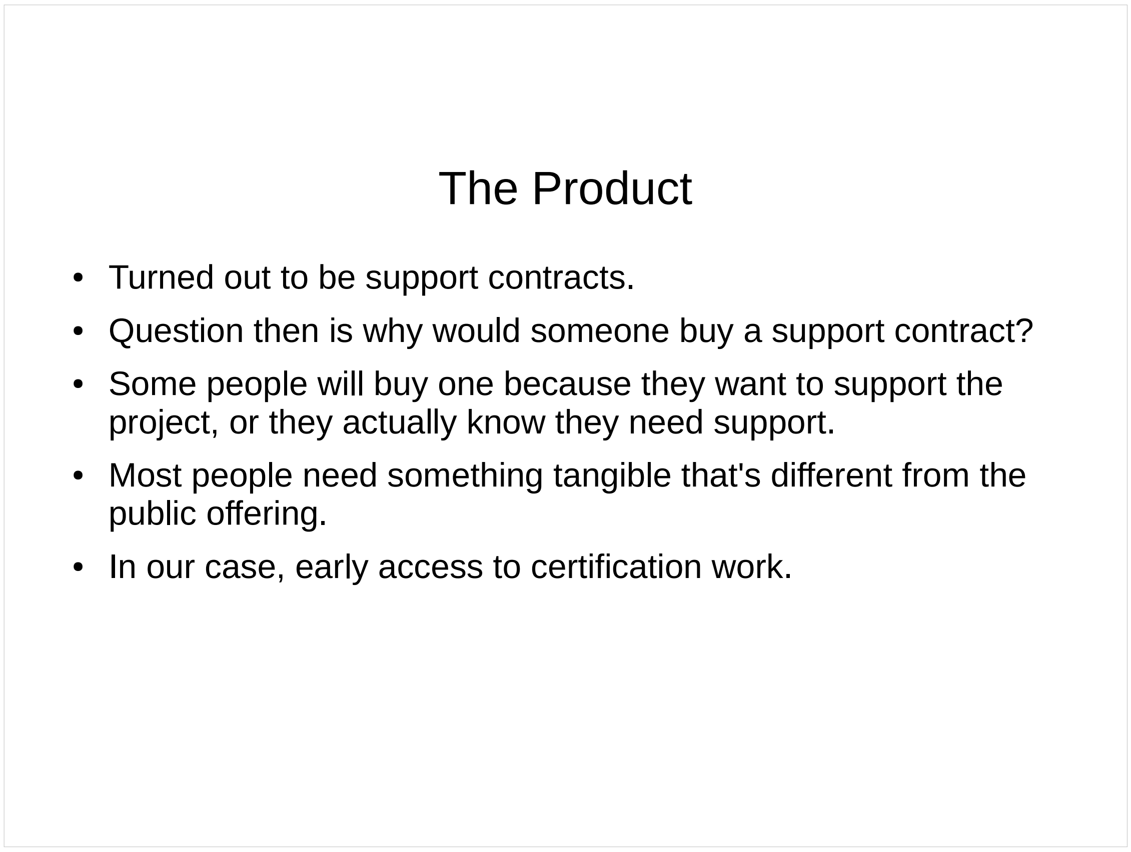
- Established a charity with ownership of the code base.
- Established a company for actual commercial work.
- Really had to find a way to make the APIs and the “product” related.
- Only accepted short-term consulting targeted to the APIs.
- Started selling support contracts.

The charity helps as it protects the project from anything stupid we might do in the commercial company. While people can actually be employed by the charity, in this case there would be an overlap between the trustees of the charity and the employees – there is an obvious conflict of interest.

Having a separate company was a solution to this.

Being “fussy” about contracts did help us stay on track, did occasionally lead to pauses in income as well though.

Support contracts slowly started to build up.



The Product

- Turned out to be support contracts.
- Question then is why would someone buy a support contract?
- Some people will buy one because they want to support the project, or they actually know they need support.
- Most people need something tangible that's different from the public offering.
- In our case, early access to certification work.

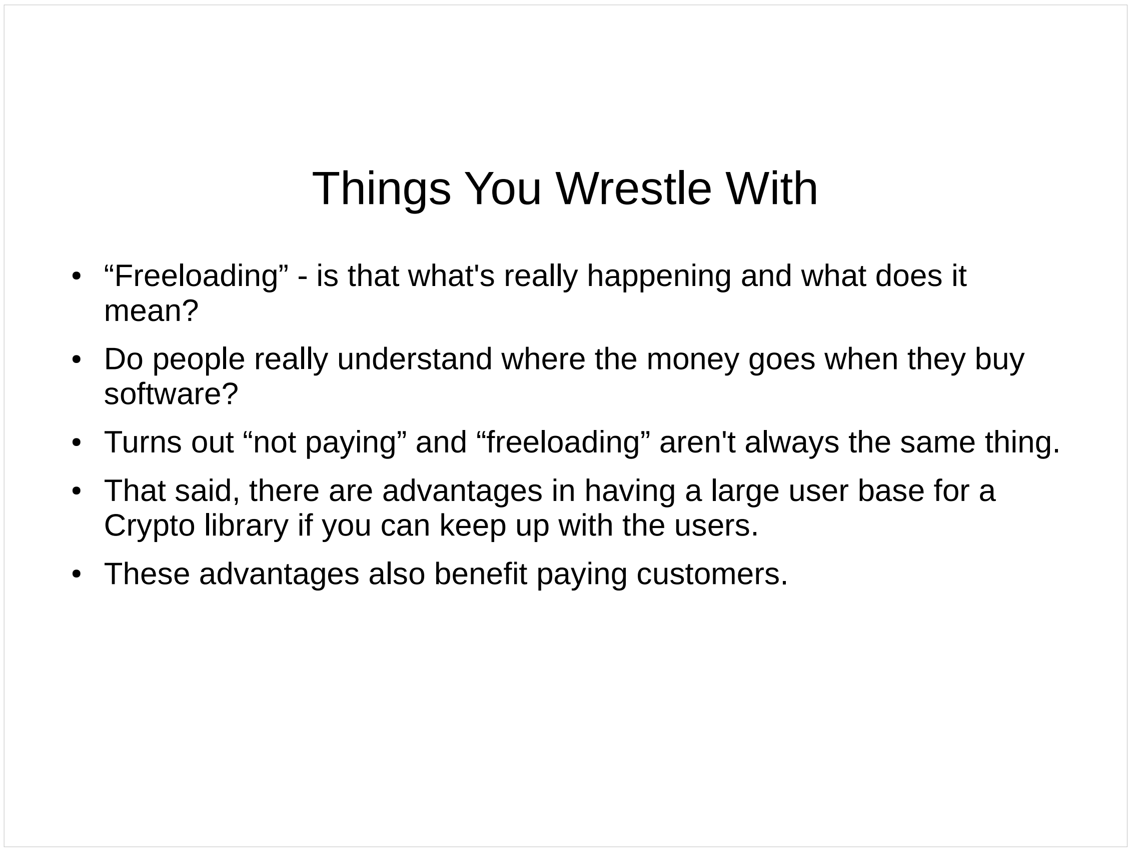
Support contracts worked, well so far. People paying for support generally want help with the APIs, and they're serious about using them. As a user base they are good to work with as it does improve the quality and usability of the APIs. The other improvement is unlike general consulting, we were finally getting paid to work on what we needed to.

Bundled consulting time into support as well, with the proviso that unused consulting time would be contributed back to the project. Apart from the “feel good” aspect, it also gave us a way of budgeting hours available.

Still a bit of a hard sell.

While all this was going on we were still looking for sponsors to help get our FIPS project off the ground. Little by little it took shape with various companies such as Orion Health, Galois, and JScape supporting bits of development. Finally Tripwire agreed to sponsor the last bit of specific work and the lab fees. We started an early access program to pay for extras and keep the project funded.

Early and on-going access to the FIPS work turned out to provide a tangible benefit people could really see and appreciate. It “made sense”. It has reduced the cost burden on sponsors and improved the quality of the product. It has also kept us alive.



Things You Wrestle With

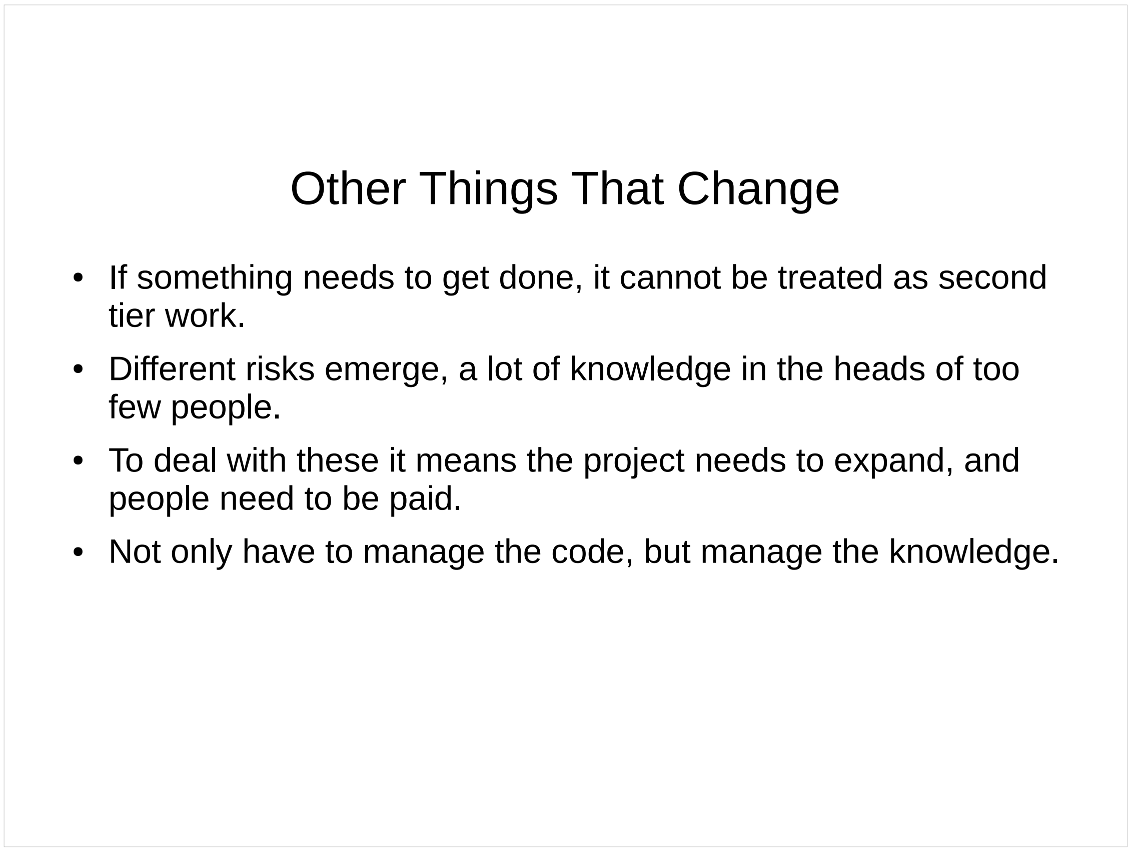
- “Freeloading” - is that what's really happening and what does it mean?
- Do people really understand where the money goes when they buy software?
- Turns out “not paying” and “freeloading” aren't always the same thing.
- That said, there are advantages in having a large user base for a Crypto library if you can keep up with the users.
- These advantages also benefit paying customers.

The “buying public” - it doesn't seem that many people appreciate how little of an actual purchase price goes on paying for on-going development. Crypto development takes place in an arms race, on-going development is everything.

Sometimes people would like to pay, but they're just as broke as you are!

Broke users are often working pet projects themselves, this often means they're doing something a bit edgy, and may do things that are totally unexpected. Feedback from people like this can be invaluable.

Finally, while there's a “market advantage” to having FIPS, it's not going to be a client's “market advantage”. With or without FIPS, a product that stinks still smells the same.



Other Things That Change

- If something needs to get done, it cannot be treated as second tier work.
- Different risks emerge, a lot of knowledge in the heads of too few people.
- To deal with these it means the project needs to expand, and people need to be paid.
- Not only have to manage the code, but manage the knowledge.

It's no longer a hobby project.

Before if someone wanted to get involved on a more long term basis, you might just pass on the odd task and see how they go. Difficult to do that when the task list is full of things people are paying to get done.

You can no longer wait for things to be completed, likewise it's not really reasonable to put an unpaid volunteer under that kind of pressure.

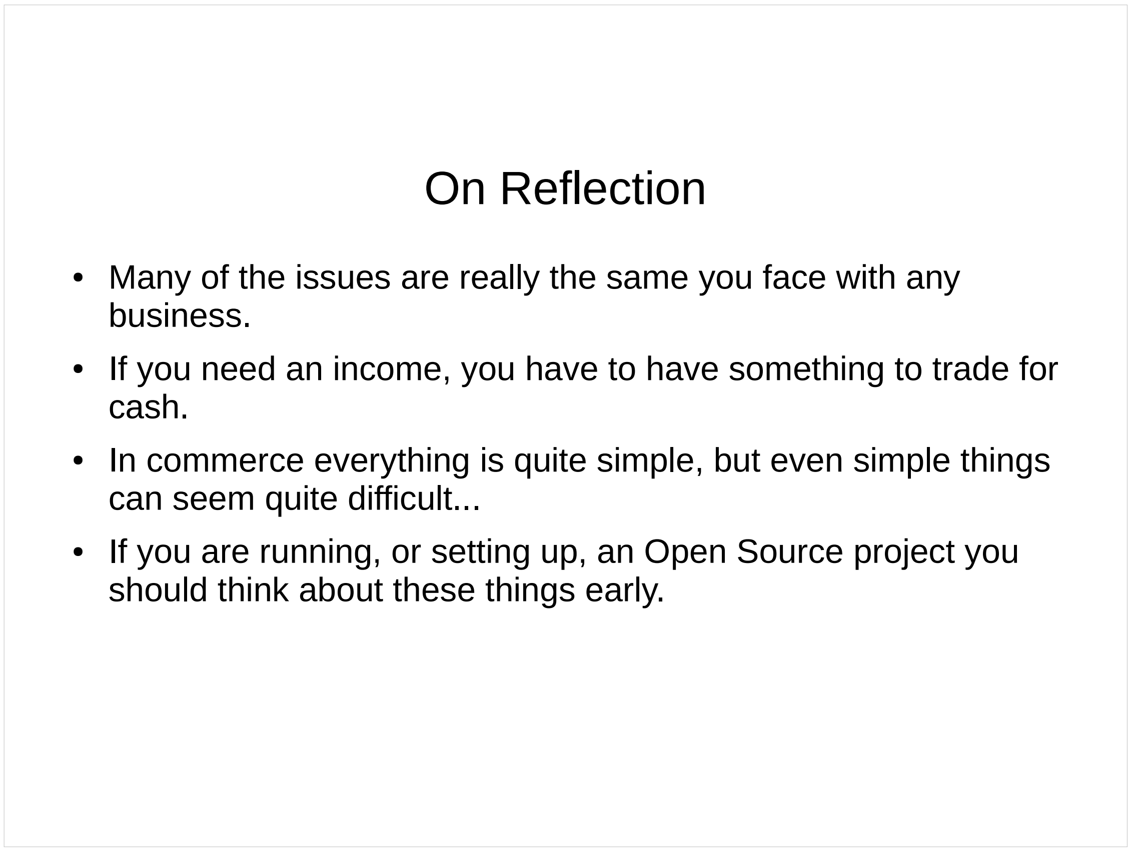
The next thing you notice as you try to get through the task list efficiently is that you've got yourself into the position where only one person actually knows how particular parts of the APIs work.

It's not just the code base we need to preserve!

So it really is expand or die!

Need to find a way to bring in more people, share the knowledge and the effort, at the same time making sure the newcomers can get paid something for their efforts.

At this point you'll realise you have turned into a real business. Oh dear!



On Reflection

- Many of the issues are really the same you face with any business.
- If you need an income, you have to have something to trade for cash.
- In commerce everything is quite simple, but even simple things can seem quite difficult...
- If you are running, or setting up, an Open Source project you should think about these things early.

As “the books” all say, you have to know your market.

You also have to know what works for you.

It's not just your software, it's also your life!

Thanks for listening.

Any questions?

It has been quite a journey and it's definitely still not over!