Every engineer’s checklist for justifying free software

Free software is not just about “no license fees”!

Malcolm D. Spence

In a few years viewing source code within the major components of software infrastructure will probably be a routine way of doing business. In the meantime it seems that the only reason managers want free software is because it is free (as in free of costs). That’s not a good reason in itself: in the long run there are compelling reasons that robust, mission critical infrastructure software should be made free software.

For over 5 years, we at OCI have been supporting free software CORBA products. Clients have been using them to build elaborate integrated software systems. During that time many of our clients have extolled the virtues of free software in regard to meeting their needs. The money to pay for software never came out of their pockets so clearly those virtues didn’t relate to price.

Free software slows ever-present market pressure to support only a single platform, or a very narrow set of platforms

It turns out the focus was on them being able to do a better job. If you have been wondering what the impetus is, for switching to free software, from a technical standpoint, perhaps the list we (OCI) have compiled over those years can help. The list breaks into fourteen categories but sometimes the benefits spill into other areas.

Configuration Options and Management

The focus here is on platforms used, compilers supported, etc.

Compiler or IDE choices: You have the source code. This means you can choose your compiler. Your ability to write highly portable code improves when you can create a development environment with a compiler that spans many platforms. Many proprietary products lock you into a single compiler, often linked to the hardware platform. Free software products often support multiple compilers that you can choose from. Keeping code portable keeps your options open.

Upgrade on your terms: If for some reason you do not want to upgrade the platform’s operating system, in lockstep with everyone else, you don’t have to. You can keep using previous versions and add patches or enhancements selectively. But be warned! Free software projects are often cutting edge. Later versions of the product may require a later version of the compiler for instance, as they stress the compiler support of programming techniques.

An obscure platform need: If you have a special platform that you want the software to run on, then do the port yourself, and create an affinity group within the community to help you spread the load on your maintenance activities. Don’t let someone else dictate platform policy for your organization. You don’t have to follow the crowd.

Are you a vendor who is feeling left out? If you are a platform vendor and you find your platform is no longer supported in a technology area, or you think it’s poorly supported, find a free software solution and support it or sponsor it yourself! Give your customer base insight into how to tune the software to maximize performance. No longer will you have to worry about a software vendor having platform architectural biases (suspected or real!), or casting your platform in poor light.

In general free software slows ever-present market pressure to support only a single platform, or a very narrow set of platforms.

Free software is an inclusionary style for platforms, versus the more prevalent exclusionary style (i.e. support for only popular platforms), traditionally adopted by software vendors. The result is: free software supports choice and proprietary systems can effectively limit choice. A platform vendor, with a well integrated stack of free software can field a competitive solution, even in a niche market.

The result is: free software supports choice and proprietary systems can effectively limit choice

Revision Management and Product Evolution

Worried about vendor viability? You don't need to negotiate putting code in escrow as insurance for critical projects. You have the source code now, and forever. Normally you can't inspect the escrowed code ahead of time. When the time comes, and you have a need to gain access to the source, it won't be the time to find out how much effort is needed to support it.

Worried about new releases? You can monitor the development activities in the beta code base, bug lists, etc. and measure progress for yourself. No more blind acceptance of a vendor's optimistic delivery dates, that are unlikely to be met. You can even help to pull in schedules, with people or funding.

Want to be a beta tester? Everyone can beta test the next release. It's an open process, not restricted to a privileged few. You can verify its stability and features, get a jump on using it, and then plan accordingly.

It's then released for development, or production use, when the free software development team feels it's ready. It's not just released so that quarterly revenue targets can be met for management. There's less chance of you helping a vendor debug their product when it's not really ready for prime time.

Enhancements

Need a feature? You don't have to wait for the vendor to add features you need. If it is urgent, you can do it yourself. Be warned though you should offer the feature back to the community. You should avoid supporting a specialized version of the code, if you can. Leverage the community.

Test a feature. When you add features and submit them back to the community, a lot of people you don't know, and don't have to pay (but who are very smart) will help you improve it.

Community members can't make bad ideas become good ideas, but they can turn good ideas into great ideas

They can't make bad ideas become good ideas, but they can turn good ideas into great ideas.

You can buy influence! If it's really important, you can also participate in the overall development process. You can influence schedules and priorities by contributing. It's the gift that keeps on giving, as others pick up supporting the project where you leave off.

If it's not that important to add new features, you can still enjoy the benefits. Nobody makes you contribute: use it "as is".

Found a bug? You can patch the code yourself if you need it right away, and post it to the newsgroups or core team to get quick feedback.

You don't have to upgrade if you don't want to. You have the source to support yourself at any release level that you choose.

You don't pay for enhancements that you don't necessarily want. You don't pay, period! No more marketing driven gimmicks masquerading as major features.

Debugging

Don't guess where the problems are: with the source available you can use a debugger to navigate, as your own code interacts with that of the product. In this way you can isolate bugs, both in your code and the product code, faster.

Developers consider the access to the source code during debugging to be a huge benefit. The improved productivity is dramatic and substantial enough in itself to warrant consideration for free software as major element of the risk reduction strategy for any major project.

Better comprehension: While stepping through the product code using a debugger, greater understanding can be gained into how the product works. Your code can cooperate with the free software product more effectively when you understand its behavior. You might also pick up some good tips and techniques. Often you are looking at world-class code.

Testing

Faster hardening of code: with access to the source code, people using a wide variety of platforms, operating systems, and compiler combinations can compile, link, and run your code additions on their systems to test for portability. This sort of scale, and parallel debug, cannot be easily duplicated within the confines of a single vendor's testing labs. Testing is the one area of software development that lends itself to scaling.

Testing is the one area of software development that lends itself to scaling

In general, developing for, and testing across, multiple platforms results in robust code. Quirks can be exposed and expelled if possible, or at least isolated.

Documentation

People outside of the core development group are more likely to contribute additional user documentation. Engineers often document something for their own needs and sense it might have value to the community and so add it to the code base. Different perspectives can provide novel solutions. It's not unusual for a few tutorial slides to snowball into a fully-fledged self-paced training class.

Code Usability Issues

It's easier to gain a deeper insight into the behavior of the software, by inspecting the source code, than it is by guessing, or trying to use reverse engineering techniques. (Which may even be considered illegal, by some interpretations of the patent and copyright laws.)

Users are now stakeholders: everyone must succeed or no one succeeds

Really interested users, also known as "Power Users" are more likely to exercise the code during beta activities, as opposed to only working with the product after it's been released. This means there are more opportunities (when the code is still in fluid state) for the user's ideas to be incorporated into the code.

Ownership

Users are now stakeholders: everyone must succeed or no one succeeds.

There is no adversarial relationship between vendor and client. That isn't to say there is no clash of ideas. Anyone who has monitored free software project newsgroups can testify to the lively discussions they contain. However the focus on the product, the openness, the peer review, all make for a good, fast, and candid way get ideas and opinions on the table. Participants learn very quickly, how to present ideas in a clear and coherent fashion in a civilized manner, according dignity to others.

Third-party tools

By having access to the source code, even proprietary tool developers are better able to provide additional tools and add-on products that can enhance the code functionality. Without access to the source code developers must frequently reverse-engineer file formats and APIs.

The tool and add-on environment is on more of a level playing field.

Evaluations

Evaluating free software is easy: no time limit pressures, no salesman calling every week.

Evaluating free software is easy: no time limit pressures, no salesman calling every week

There are no "limited use" rules, restricting the number of evaluators or product features: everyone in your organization, who is interested, can get involved.

There's no legal paperwork to process, or "permissions" to seek. Only the evaluation consumes your time. Informal evaluations are easier. (Do it at home if you feel strong enough about the potential of a free software solution.)

A free software evaluation activity helps you sort your needs, priorities, and get some familiarity with the domain of both the problem and potential solution. If you want to speak with a vendor subsequently, about a proprietary product, you can have that dialogue from a position of strength. You are now an informed buyer!

Benchmarking

Benchmarking is an important activity for determining if a product is right for you.

Remember that many proprietary vendors make you sign agreements that prevent you from publishing or sharing benchmarks regarding their products - particularly comparisons that you might make between their product and other vendors' products. It's an understandably protective position: bad benchmarks can get out, sully the reputation of a product and be expensive to recover from. However no such requirements come from free software vendors! Free software communities have no interest in suing their stakeholders and partners. They're the ones who can help the community in improving the code. The community isn't interested in hiding the results or inhibiting evaluations.

Get those benchmarks out there in the open and let the community assist you in building the right kind of benchmark. Share or improve what's already out there. Free software providers will usually package suggested benchmarks. Build on what's available, this will save you time. Add your benchmark code to the mix. Someone else is likely to improve on it for you. The community will also help you interpret them.

You must measure what matters. You cannot improve what you cannot measure. The more measuring goes on, the more improvement will occur.

Porting someone else's benchmark to your platform and comparing results helps us all understand implementation differences across platforms. This is an important insight for the evaluation process. Does a product mask or magnify the variation that occurs between platforms?

When done right, benchmarking can cost a lot of resources. Free software approaches can provide you with more leverage.

When done right, benchmarking can cost a lot of resources. Free software approaches can provide you with more leverage

Open approaches can help avoid errors in the selection process. Many free software projects include performance testing as part of their regression testing. Performance issues are put in the open to be addressed.

Proprietary vendor benchmarks are often very selective. They have to portray their products in their best light. Free software benchmarks play to the market of free ideas.

Security

Free software means what you see is what you get. You can inspect the code, line by line, to ensure that no disgruntled programmer has buried logic bombs, trapdoors, Trojan horses, viruses or any other nasty surprises in the code.

You don't have to worry that being late with that license fee might result in a locked up system. There are no worries, as with proprietary systems, that the code may contain the means to disable the software, and effectively your business. Free software is not UCITA!

You don't have to worry that the weak link in a security strategy might be some proprietary application with poor defensive measures. You can add your security features to free software, if you wish, and ensure a consistent level of protection across all applications in the system.

Licensing

With free software there are no licensing fees, no development fees, and no runtime fees.

You can put free software where you want it, when you want it. Performance and other considerations drive those decisions, not the licensing model constraints (such as node locks).

There is no arguing with management about money for license upgrade funding just to stay current; upgrading is now mostly a technical decision, not just a financial one.

Are you a VAR? The cost model for your products is more predictable: no more sweating about vendor licensing model changes, which might break your pricing strategy.

There's no vendor trying to use a combination of licensing and proprietary extensions to keep you locked into an implementation.

There are no concerns that your vendor might disappear without a trace leaving you with node locked software, a rollout coming up, and no way to implement more licenses (in other words, you have no way to deploy your application). This may seem remote, but it does happen and has catastrophic results.

Switching costs are not inhibited, or influenced, because of their magnification due to the sheer volume of systems involved.

You can recommend the software to others on its own merits, without worrying about the cost implications.

You can build it into, or ship it with your products, as a way to help your customers, and to improve your product's utility, without affecting your pricing edge (cost of goods sold).

Your own product's functionality can continue to be improved and take advantage of the free software product's improvements without worrying about the repercussions of your customers having to pay upgrade fees. They can stay current with minimal financial impact, or you can ship the new version with your product.

Free software levels the playing field for those smaller companies who cannot negotiate site licenses and other large discount programs that often give larger companies an additional pricing edge.

The special impact of free software on middleware

At OCI, we elected to support free software middleware. We did this for many reasons: we think free software and middleware is a particularly good fit. It facilitates open systems and promotes choice. Object middleware is a special kind of software. Its utility stems from its ability to ensure consistency and interoperability amongst applications with diverse backgrounds and capabilities. The value that middleware offers includes:

- The ability to provide a standard, stable, consistent interface to a wide variety of applications, on a broad set of platforms and enable their inter-operability.
- It decouples service providers from service requesters.
- It enables parallel development of service and client.
- It hides implementation details behind standard interfaces. This allows the implementation to change without disrupting or breaking existing clients.
- It allows different types of clients to share the same services, often simultaneously.
- It frees the developers of distributed systems, from the burden of developing networking software, allowing them to focus on their own application's particular needs.
- It enables the migration of services to new platforms, increasingly diverse and specialized communications technologies, operating systems, and implementation languages.
- It allows legacy systems to be "wrapped" in standard interfaces, giving them the ability to become distributed components as well.
- It allows the co-existence of solutions employing multiple languages.
- It protects the existing legacy systems, and yet future proofs what you develop today against language, platform and communications obsolescence.

Vendors of proprietary products are typically under pressure to differentiate their product with extensions. These value-added features can often lead to incompatibility and confusion about portability and interoperability. Vendors must balance standards compliance with maintaining an edge. This is not an easy task: a product that is completely standards compliant can be more easily replaced. To protect market share, middleware vendors must selectively add proprietary new features along with standard ones to maintain a hold (sometimes termed a compelling value proposition) on their client-base.

Free software is free from these pressures. Often, as second-generation products, their value proposition is that they support the standards (where they exist), very closely. This is their way to differentiate themselves from the other products, already present in the market place. They hope to use the consensus achieved in that standards community, and the experiences derived from multiple product implementations by users, as a way to stabilize the technology domain. Then from that technology create a commodity item and thus another stable building block in the technology layers that make up distributed systems. Users and vendors can then move on to other areas, along the value chain, higher up the ladder of abstraction, to advance technology and create new value.

Applications are the true value-added software. Middleware should be "low impedance" software for enabling application interoperability and systems diversity. Software historically has moved towards a monopoly, as a way to achieve easy interoperability, via uniformity. Within the middleware market there should be enough diversity to foster innovation and yet with sufficient uniformity to enable cooperation. Free software can act as a break in the natural progression towards a single dominant vendor. This progression towards a monopoly is not healthy for systems. It is termed pejoratively, "monoculture", because of its vulnerability to attacks by viruses etc.

In free software the users contribute capabilities to the product, which they want to have available. This might include extensions beyond the standard but they are created by a user's need, not by a vendor's motivation to lock in. These activities can become the basis for a standards submission as they can prove the viability of an approach and enable many users to validate their utility in real life situations.

Conclusion

In the past, standards emerged towards the end of product and technology cycles. Often they were too late to do any real good: they really just codified past technologies. Newer standards, that try to get out in front, often lack the practical experience that really enables them to gain traction. Free software can offer the best of both worlds. A free software project can be a work-in-progress, upon which standards can be based, practically, and gives no feeling that the software is locking you in.

Early APIs can coexist with later ones, thus users can migrate over time, at their pace.

Openness is as much about supporting diversity, as it is anything else. Middleware is a powerful leverage point for ensuring diversity can be accommodated.

Linux on the desktop is part of, but by no means the whole story, for free software.

If you're not convinced, think on this: being an active contributor with a free software project offers you visibility, sharpens your skills, looks good on your resume, and enables you to participate in a virtual world-class team. It can give your present job an added edge.

Copyright information

About the author

Malcolm Spence has a Diploma in Aeronautical Engineering from Kingston upon Hull College of Technology, did post graduate study at the College of Aeronautics, Cranfield, (both in England), and then obtained his MSC in Civil Engineering (structures) at UMR. He also attended jet engine school at Rolls Royce (UK) and the Tuck Executive Management course at Dartmouth. Malcolm has directed the free software line of business at Object Computing for the last five years. Prior to that, for a year, he was a freelance marketing consultant with various start-up companies. From 1981 to 1998 he was with St Louis sales office, and then the industry marketing group, at Digital Equipment Corporation in Boston. His later assignments involved providing marketing support for DEC Central Engineering OO initiatives in the Asia Pacific region. Before joining DEC, for thirteen years, he was an aeronautical engineer with the McDonnell Aircraft Division of McDonnell Douglas in St. Louis. He resides in the St. Louis area with his wife and three children.