Custom Software Development in Post Production

Andrew Chapman
Framestore CFC

Jack Brooks
Walt Disney Feature Animation

David Hart
PDI/Dreamworks

Daniel Maskit
Digital Domain

Steve Sullivan
ILM

1 Panel Topic – Andrew Chapman

Most post-production and digital effects work these days employs custom software to varying degrees. This software may be a necessity for the high-end work, and it mostly gets the job done, but from the perspective of the artists and other users, it is often poorly written, hard to use, and causes delays and frustration near deadlines when it can be least afforded.

This panel consists of software developers and artists who have experience in creating and/or using such software and will discuss in broad terms what is wrong with it, why it is this way, and how it can be improved.

There seems to be a willingness to accept unreliable and poorly written software in our field because the average technical ability of the end users is generally much higher than in other industries. While it is true that many users are able to work around the problems they find with the software, it is still costing us greatly in wasted time and effort. It also contributes to the overall feeling that CG production is harder and more frustrating than it need be, resulting in longer hours and a less pleasurable experience for everyone.

Our industry is maturing, with a larger and more consistent pool of work. However, software development is still generally done in a fairly amateurish fashion. It is not enough that a piece of software works on one particular project. It needs to then be available as an easily accessible tool for doing similar jobs in the future. We need to provide a more sustainable foundation of tools, rather than simply starting from scratch with each new project.

Open Source software is an interesting avenue for improvement, as the nature of development in our field makes this model particularly suitable. Tools can be quickly written to get the immediate job done and then passed out to the Open Source community for the robustness and usability features they are often lacking. By the time they are needed again by the originating company, they may have improved greatly with no effort or expense on their part.

Making custom software Open Source also helps to alleviate the problems of using proprietary tools when a large proportion of the users are highly migratory. Users are often unwilling to dedicate themselves to learning or helping to improve custom software when they know they might soon be working elsewhere and those tools will be unavailable to them.

However, if the tools were made Open Source then people will be encouraged to contribute, as they are using and building on something they can use again in the future, and the skills and experience with those tools will be appreciated outside their current employer.

2 Position Statement – Jack Brooks

As, Director, Technology at Walt Disney Feature Animation, I have one of the larger custom development groups in the business. We deploy a wide range of 3rd Party, completely custom, and extensions to 3rd party software products. In the six years I have run the software development team here, I have struggled extensively with the issues and concerns expressed in this panel problem statement. We have evolved from a studio with almost all of the core software being custom, to a place where we have a broad framework the supports a wide range of internal and 3rd party components. I strongly believe that custom development has a critical role for high-end production, but it must always be looked at as cost value trade off.

The answer to the question of custom development or not is also largely driven by the size of the studio/project. The smaller the project or studio less sense it makes to write a custom solution, and vice versa. This is driven by the fact that ultimately the decision is about cost. Any look can be created, in the most extreme case, painted frame by frame, with existing software. The question is, will it more economical to write a solution than to brute force the solution. The larger the project, the more likely it is that the production savings will exceed the development costs (even small savings add up across 1000 shots).

Many of the issues described in the problem statement can be traced to how projects are managed. The most critical change we made to address this was insuring that key production users took a very active role in the development. We also had to change how we measured the products success. Custom development is typically attacking problems that we have not solved before. Thus a traditional requirements phase is not fruitful as most of the time is spent on the wrong items. Instead we focus on some high-level objectives and target dates and then measure the progress by evaluating whether we are getting more value from the project than we are investing. Success is not determined by whether we

met the original (often flawed) goals or created a robust software product. Both are good things if they happened, but, we are not in the software development business so the only measure is whether this development enabled us to do more for less.

While our process may appear amateurish to developers in other industries I think that there is the right amount of structure for our environment. Related to this is the perception that the first time we create a product it should be the ultimate solution. I feel strongly that this is not a reasonable burden to place when developing something new. We consider the initial development of a tool to be a throw away after the first production use. This does not always end up being the case, but it frees the team to move quickly and meet the immediate demands. Then based on the results, we decide if we want to continue to use the tool, what it should really do, and how much it is worth investing to make it robust and general. These are not judgments that can be accurately made until it has been through it's first production.

Using an Open Source paradigm in our industry is interesting and something I would love to give more effort to, but I fear that studio pipelines vary enough that the type of community development that occurs on something like Perl is not generally likely.

3 Position Statement – David Hart

There are good reasons quick, sloppy software is written in production without the overhead of formal software methods or lots of documentation and testing. Tight production schedules, complex dependencies and quickly changing staffing are constantly at odds with proper software engineering principles. Often the mistake we really make is not writing software poorly, but expecting software that was by necessity written very quickly to automatically generalize and apply to situations other than what it was developed for. In addition production planning often underestimates the time needed for 'in production development' and instead assumes that the custom software used for a few shots will work for many more with little or no changes.

At PDI/Dreamworks, the key factor determining whether a piece of software will be high quality and general enough for widespread use is whether or not it falls within the production budget. We use two different development methodologies depending on whether the software is intended for long or short term use. Long term software is developed and funded outside the production. It is written using formal software engineering methods, including documentation, testing and review. Short term software is developed within the production, subject to the production's scheduling and staffing requirements, funded directly by the production budget, and few coding standards apply. Having the option to choose between short and long term development modes for any particular task is wonderfully flexible. Unfortunately most CG studios, expecially smaller ones, operate wholly in this short term model out of necessity.

The kind of software engineering we learn about as CS majors is heavy and makes the assumption that the code needs to be maximally general and reusable. It also assumes that the code is the product, reinforcing the need for reusability. Our case is different, the film is the product, not the code, and the dirty tricks we use to get the film done won't break the film once it's released. Because of this and because hectic production schedules aren't going away, I think we're stuck with the short term software model for the foreseeable future. We should embrace its flexibility and work to make it better. There are specific things we can do to vastly improve the short term coding model. In terms of software engineering for CG production we need to improve, or perhaps establish, lightweight formal methods.

Open Source is indeed an interesting avenue for development, one that should be explored, and there are even some very good projects in the works out there. Open Source may address some of these problems but if CG studios want to go this route, it also brings a host of new issues to the table: ownership and intellectual property rights, how to manage software changes, release schedules and feature conflicts for different productions, just to name a few.

The onus is on CG developers as a community to make these solutions a reality. Studios are not motivated to invest time or money unless they see direct benefits for their productions, nor should they be expected to. Its our responsibility to figure out how (or whether) to reuse code that was written at a fast and furious pace without documentation, and how to write code quickly that people will want to reuse. It is our responsibility to participate in developing Open Source tools and use them to solve problems at home and work and help the studios we work for to see the direct benefits of these efforts.

4 Position Statement – Daniel Maskit

At Digital Domain, we are in the business of producing visual effects, not software. Obviously we are happy to commercialize our more successful software projects if there is a possibility to do so, but only if doing so will not compromise the competitive advantage we feel we gain through our in-house development. Whenever we identify a new software need for production, we start out by determining whether or not it makes sense to develop something ourselves. In general our options are to buy something from someone else, find some free software that meets our needs (this is quite rare),

develop a quick and dirty tool to get through a show, solve the problem with plugins, or develop a solid, general piece of software.

Given the resource constraints that we operate with, this last option is only taken for tools which will be used by many productions, can contribute to our competitive advantage, and can be developed within a production's time constraints. As much as we would like to just implement all of the cool ideas that we come up with in software, our process is driven by production. We generally only develop new software if we think it is providing us with a competitive advantage. In general this precludes our releasing software as Open Source, although we have created some widely used Open Source packages, with the best example being FLTK. We are more likely to take a smaller role in Open Source, but do try to contribute back to the community when we have modified existing packages.

For us software development is always a balancing act. On the one hand we have people who are capable of producing extremely high quality software, from an engineering viewpoint. On the other hand we have some of the most demanding users on the planet. In between those extremes we have a wide range of skills and requirements. Developing processes to allow each of these groups to find appropriate compromises to make is an art in itself. When we ask our users to be more sophisticated, we are generally giving them something in return. Usually this is a faster delivery time, but sometimes it is better integration into existing production pipelines.

And I think that the mention of this integration gets to the crux of the problem for us. No piece of software is terribly useful if it can't be fit into our workflow. When we develop code ourselves we can ensure that it plays well with our pipeline. When we use other people's software, we generally end up having to write our own code around it to make it fit in anyway.

As for the quality of software, I think that we do better than people might realize. You would think that our tools would benefit from a formal quality control process, and the use of professional testers, but it is unclear that our software is significantly worse than many of the third-party packages produced by professional organizations. And we have the added advantage that we can roll out bug fixes without going through a lengthy formal release process.

Ultimately what our clients care about is not whether we have easy to use software, but whether we are able to give them something that has never been done before. And our artists know that when the tools don't work the way they think they should, we are happy to make changes to the code.

5 Position Statement – Steve Sullivan

I'll be giving the perspective of developing studio tools and infrastructure at ILM, where we co-mingle proprietary and vendor tools freely. Someone speaking on per-show or per-shot development might have very different answers.

- Custom production software is essential to cutting edge effects. Vendor tools alone will generally not get you state-of-the-art.
- Custom production software is essential to competitive edge. Many artists rotate among studios per project, and artist experience isn't the distinguishing factor it once was. Better/faster tools/techniques/infrastructure is one clear way to differentiate.
- Software developers must recognize the project-driven nature of the industry, that wonky interfaces pose a real cost in training and productivity. In the big picture, utility trumps cool, so stick to standards unless absolutely necessary to support the next great thing.
- Post-production is a complicated process with many inputs, stages, and outputs. It seldom pays to polish any particular tool to perfection, since after, say, 20% improvement, the bottleneck has moved on. Often better to make smaller advances distributed across artist workflow.
- Quick turnaround times for custom software can be a lifesaver in production (and likewise, poorly-designed or tested code can be deadly)
- The vendor market is tough, and complete reliance on vendors is risky. In some areas, we could buy solutions a few years ago that we can no longer buy today.
- At larger shops like ILM, long-term design and serious software engineering is essential. A small number of developers hand their systems off to a large number of artists working on many shows in parallel. We cannot afford developers writing hacky per-shot solutions to be used primarily by the developers themselves.
- Open Source is not a viable solution for most tools (see the long-term design and software engineering point above). Effects studios are driven by their projects and specific needs, and Open Source progress/quality doesn't often align with them. Everyone wants to do the quick/small thing needed for their show, no one wants to do the hard architectural slogging. (e.g. FilmGimp)
- Open Source can work well as a mechanism for fostering standards (e.g. OpenEXR)