&lt;&lt; RETURN

Growing a Dedicated Tools Programming Team: From Baldur’s Gate to Star Wars: Knights of the Old Republic.

By Don Moar
Lead Tools Programmer, BioWare

1 Introduction

The tools programming team at BioWare is responsible for the content creation, asset management, build construction, end-user and localization tools on each project. This has typically amounted to anywhere from 20 – 50 individual pieces of software per project ranging in size from just a few lines for some conversion utilities to hundreds of thousands of lines for the designer toolset. While some of the tools can be reused with varying degrees of modification, there is still considerable new development required for each project.

At the beginning of 1999, the tools programming team at BioWare consisted of two programmers working on Baldur’s Gate Tales of the Sword Coast and Baldur’s Gate 2 Shadows of Amn. Currently, the team has 16 programmers working on five projects and by the end of 2004 it is expected to increase to 22 or about 30% of the entire programming department. The average turn-over rate is approximately one programmer per year; with the majority of programmers being transferred to the game and graphics teams.

This paper will discuss the issues surrounding this growth. It will explain why BioWare places such emphasis on tools development and how it has been able to expand to such a degree. In addition, this paper will describe the roles and responsibilities of the team, how the team is best used on the projects, the consequences of success and the price of failure. Finally, the paper will describe the practices used for operations, change management and development.

2 Why a Dedicated Tools Programming Team?

BioWare devotes a lot of time and energy to the continual evolution and improvement of its tools. This is money that could be spent directly on the game. As such, it is fair to ask "Does BioWare benefit from having a team of dedicated tools programmers as opposed to using whichever game programmers happen to be available?"

2.1 The Tools Programmer Advantage

Everyone relies on tools. In software terms (game development, specifically), tools are necessary to help people perform data entry and automate repetitive or error-prone tasks, improving reliability and efficiency.

If, for example, it takes one tools programmer one week (40 man-hours) to write a tool that simplifies and reduces build times by one hour and builds are performed twice each week, it will take 20 weeks before the development time has paid for itself. On a two-year project, that will save 160 man-hours in build time (about one man-month), but that is not the total benefit. Faster build times, means more QA can be done, which hopefully results in fewer problems in the final product and overall less time spent on support. Assuming that there are four QA analysts on the team, the above one hour reduction in build time results in four extra hours of QA per build. The payback time for the development effort is then reduced to four weeks and results in 800 extra hours of QA on the project.

In the roughest of terms, each tools programmer should be responsible for saving or returning to the project at least 2000 man-hours per year. If it is no more than this, then nothing has been gained. Depending upon the tool, it may be possible to amortize its development over the project's post-ship support cycle as well. That is, a tool that reduces build time will continue to be of value if and when it comes time to release an update. Again, looking at the improved build time example, with one week of work, this tools programmer returned 960 hours (160 + 800) to the project or almost half of the annual "requirement".

Note that it will not always be possible to assign values such as these to the work that is done by a tools programmer. This is because new tools may enable the content creators to take advantage of key features that were simply impractical on previous projects. In such cases, the value of the tools is essentially the value of the finished project.

2.2 Specialization and Dedication

Tools programmers should be actively finding and exploiting as many opportunities to increase the overall effectiveness of the development team as possible. The best way to accomplish this is through experience working on other projects. This implies the creation of a team of programmers hired for and dedicated to that task alone.

Originally, BioWare focused on developing a small, but highly cross-trained tools group. The idea was that any tools programmer should be able to easily migrate from content creation to asset management tools, for example. While this worked when BioWare was smaller, it became clear that this is a bad idea for the same reason that dedicated graphics programmers should not be expected to easily switch to AI or networking. That is, the strategy started to fail when the complexity of the tools was such that it required significant training time to be proficient in each area.

Eventually, BioWare realized that creating a dedicated team encourages the development of particular skill-sets which, while expensive, can be applied across multiple projects. BioWare has already identified several key areas where its tools programmers are beginning to specialize such as asset management, content creation and localization. This has started to allow the solutions to problems in these areas on one project to be more easily carried forward to the next.

3 Staffing the Team

This is the most fundamental part of building a tools programming team. If the team is not staffed properly, nothing else will matter. The key is to hire programmers with the right skills, who want to write tools and then keep them there.

3.1 Finding and Hiring the Right People

Hiring programmers who want to write game code, AI or graphics onto a tools team is a waste of effort. Eventually, these programmers will become unhappy and unproductive at which point they either must be re-assigned or will likely quit altogether. In the end it does not matter what happens to these programmers after they leave the team because all the training time spent on them and experience they have gained will be lost.

During BioWare's first major round of hiring for the tools team, in the spring of 2000, this distinction was not made. In fact, it was just the opposite. The approach was to deliberately hire programmers onto the tools team as a probationary exercise. After six months or a year, these programmers would be 'promoted' off the tools team onto the game or graphics teams if they were working out. This was a terrible idea. Out of the five programmers hired in that round only two remained by the end of the first year. The other three had been transferred to game programming teams either at their request or the request of the lead programmer on that game.

Tools programming is already perceived as being somewhat less glamorous than the graphics and AI disciplines. Perpetuating the notion that tools programming is a role out of which people are 'promoted' will not help attract the talent necessary to create first-class software. Tools programming must be thought of as a valuable long-term career option into which only the best candidates will be accepted.

Candidates during subsequent hiring rounds were hired explicitly because of their desire and passion for writing tools. This attitude has cost BioWare some potential new employees but all that really means is that it eliminated the subsequent round of hiring that would otherwise have been necessary. Consequently, the average turn-over on the team is now very low (approximately one programmer per year).

3.2 Skills

The kinds of skills a tools programmer needs depend upon the kinds of tools required to build the next project. Without experience on previous projects, guidance in this area must come from the project's leads and producer. As experience is gained, the tools programmers themselves should be able to help drive the list of skills required to increase the team's value to the project and company.

BioWare's games are typically, extremely design-heavy (i.e. contain a large volume of non-art content), contain tens of thousands of resources and have hundreds of thousands of words. This requires programmers specializing in, among other things, Windows-based application development, user interface design and databases. Excellent communication skills are also critical at BioWare.

In spite of this focus on Windows and database programming, there is still a need to have tools programmers who can build art tools. Until recently, BioWare's tools team has lacked specific expertise in that area. This gap developed during Neverwinter Nights as the company shifted from making 2D to 3D games and widened on Star Wars: Knights of the Old Republic. It is only now, on Jade Empire, that this gap has started to close as the tools programming team began to actively recruit tools programmers for that purpose.

3.3 Keeping the Team Together

Getting the right programmers on to the team is not enough. Even the most committed programmers may lose interest if they are not satisfied with their role on the team. Satisfaction typically arises from some combination of challenge, contribution, growth and compensation. The exact proportions depend upon the particular programmer in question and will have to be worked out in detail with each one either on an ad hoc basis or during more formal reviews.

The need for a challenging work environment where people feel as though they are contributing to the success of the project applies to almost all game developers and programmers, tools programmers specifically, are no different. Compensation comes in many forms: a comfortable salary, a good benefits package, bonuses and a fun place to work are all part of it. Again, by providing a solid overall system of compensation for the entire company it will, by definition, apply to the tools programmers.

Growth is a unique issue when it comes to tools programming. In order to really combat the perception that these positions are where entry-level game programmers start, a career path must be set up that offers real long-term opportunities. At BioWare, tools programmers have the option to specialize in a particular area such as asset management, content creation or one of the other areas identified earlier. Starting out on a project helping a more senior tools programmer in a particular area, the tools programmer may take on full responsibility for this area on a subsequent project. Once a level of expertise in that area has been acquired, that tools programmer may choose to further specialize in that area, select another area of specialization or take on more leadership and management tasks. In reality, it usually involves a combination of all three.

When BioWare (and the tools programming team) was smaller, all the tools programmers were expected to be able to work on tasks from any project. This and their relative inexperience made it impractical to divide the group up amongst the different projects. Instead, the tools programmers were located in a generally centralized area in the office space. One benefit to this was the development of a strong group bond. This bond has proved invaluable for the communication and adoption of the group's roles and responsibilities (mentioned below). This sense of group identity helped to foster collective ownership of all the tools across all the projects.

As the team has grown and the number of senior tools programmers has increased, it is now possible to assign groups of tools programmers to projects for extended periods and locate them in the project area. This has resulted in improved communication between the tools programmers and their customers (artists, designers, programmers, producers and QA analysts). The benefit to this is better and more useful tools that meet the needs of the specific project which, in turn, should produce higher quality and more successful products.

Which is a better organization? Assignment and close proximity to their assigned project should produce a bigger short-term benefit for the project and company. On the other hand, group association means higher quality tools across all the projects and should produce a bigger long-term benefit for the company. They both have their place depending upon the size of the company, the size and experience of the tools programmers and the number of projects currently under way. No matter which model is followed, be careful to still take the time to build that group bond.

4 Roles and Responsibilities

How does a tools programming team fit into the rest of the project or company? This has been an evolutionary process at BioWare; the tools programming team growing and adapting as the company continues to change. What follows is a description of the key points around which the tools team at BioWare has currently structured itself.

4.1 Communication

Tools programmers are no different than anybody else on the project team; their ultimate focus should be on the end product, the game itself. This should not be a surprise.

Unlike the rest of the team however, the primary customers of the tools programmers are in the next office or just down the hall. The level of satisfaction the team has with the quality of the tools and responsiveness of the tools programmers will have a direct impact on the final game. Good communication is the key to that satisfaction and that is why it is such a vital skill for tools programmers (actually, everyone needs this skill).

There are lots of ways to maintain or improve communication. Put those who need to communicate in closer proximity, conduct regular meetings and support informal or "extra-curricular" gatherings are just a few possibilities. What will work depends largely on the corporate culture, work environment and the current state of the project. Watch for signs of communication breakdowns and deal with them immediately.

To facilitate better communication, the tools programmers at BioWare have a primary contact among the artists, designers and other customers for each major system. The tools programmers and their contacts meet fairly frequently to discuss what changes (if any) are coming up in the corresponding system. The contact's responsibilities are to help prioritize the feature requests that come in from the other users, communicate when the tools programmers feel the features will be ready back to the users and validate the new features (primarily the UI) before they are released. Getting feedback on that user interface early is critical. Do not let the content creators put this evaluation off "until they have time" because that will not be until they actually need the tool (i.e. when it is too late).

One thing to note: the most effective tools become a natural extension of whatever process you are following and do not impede it. The users should be able to forget about using the tool completely and be free to concentrate on their work. One consequence of this is that unless there is a problem, no one will really care about the tools that are used on the project, no one except the tools programmers. This means that tools programmers are likely to encounter only two kinds of comments about their work from the users: it is buggy or it is late. Close proximity and reliable communication can go a long way to increase the compliment to complaint ratio.

4.2 Reduce Development Time

The role of any tool (in either software or physical terms) is to make the job easier or the output better. The result of either is a corresponding reduction in the time necessary to complete development on the project. This means that there is more QA that can be applied to the product and / or that the product will be delivered sooner. Overall, reducing development time must be a consideration in all areas of tools development, from the performance of an algorithm to the number of clicks in a user interface required to get the job done.

Reducing development time involves three fundamental principles that must be kept in mind when writing tools: First, usability is king (or queen) with the caveat that personal productivity should not come at the expense of the team’s productivity. Second, tools software should try to catch errors at the earliest possible moment. Finally, a planned and scheduled delay is less expensive than an unexpected outage.

One example would be a request to add validation checks before the content creators can post their changes to the production environment (the checks would be optional for local testing). The production environment is the set of data and software that comprises the current game and its corresponding construction tools. This is where everyone goes to get the material required to do their work and advance the development of the game. This request arises from the fact that it takes more time and people to track down and fix content problems after they have been submitted than if they were caught earlier. In some cases problem content can actually block other content creators.

For the purposes of this example, assume that these validation checks add between 2 and 20 minutes to the submission time depending on the size of the content. Also assume that the checks cannot be made any faster. Now, adding these checks will slow down the introduction of content which reduces individual productivity but should stabilize the entire build and increase the team’s overall productivity. By not enforcing these checks on the local environment, the content creators still have a very usable system.

The example above, demonstrates the first two principles. The last principle suggests taking the time to do things right even if it results in a bit of a delay because in the end it will actually save time. This is really just an extension of the first two principles, considering the team as a whole and catching errors before they have an opportunity to cause a problem. Think of this as a productivity enhancement for the tools programmers; typically, they can use all the help they can get.

4.3 Protect the Data

Obviously, if a software tool does not manage to reliably create, edit or otherwise manipulate its data without damaging it, it is not going to reduce development time. In addition to the time needed to re-implement or regenerate what was lost, tools failures will cost the project the time spent waiting for the program to be fixed. This actually costs the project on several levels as the users can not proceed with their work and the tools programmers are not working on something of more value.

Protecting the data starts with using solid engineering and change management practices during the development of the tools themselves. It also involves preventing the uncontrolled introduction of new and untested tools software into the production environment. The challenge is in convincing the rest of the project that this is necessary and workable. Fortunately, the easiest way to get buy-in from the rest of the team is to do the exact opposite for a project or two, although that, too, offers a certain amount of pain.

Between 1999 and 2002, BioWare used rather ad hoc methods for tools development. New software builds would be made and released on demand, usually without any more testing than was required to see if the requested feature worked in a single case (and that was often insufficient). Application crashes were frequent and generally accompanied by some degree of data loss. Unfortunately, because there were so few tools programmers and their workload was so high, it was thought that proper design, documentation and testing were simply luxuries that could not be afforded. This led to significant tools quality problems up to the end of Neverwinter Nights and part way into Star Wars: Knights of the Old Republic.

This chaotic approach to tools development meant that by the end of Neverwinter Nights, new toolset builds had to go through a two day verification cycle in QA before being released to the design team. Often, the QA cycle would be incomplete when another new build became available. As getting the latest build out as quickly as possible was of paramount importance, QA on the current build would have to stop and the entire cycle would start over again with the new build. As a result it could take anywhere from three to five days for a particular change to get into production use.

It was clear that something had to change. Starting in early 2002, changes to tools software were planned and coordinated with the team, then rolled out into the production environment on a regular, weekly, schedule. This appeared to slow things down at first, but the critical bug rate started to fall and more time could be spent adding the required features. The builds were not coming out much faster, but they were coming out much more stable and reliable.

4.4 Tools not Systems

Focus tools development effort on creating the custom software necessary to complete the game construction pipeline. Do not waste time writing software that can be otherwise purchased from another company. Unless there are extenuating circumstances such as the software available is too expensive or does not have the complete feature set required, take advantage of the expertise these other companies possess. Remember that any time spent developing an alternative solution to a commercial product is time taken directly from improving the project.

During Neverwinter Nights, the tools programming team at BioWare was asked to write a replacement for the current bug-tracking system. The bug-tracking system that was in use at that time had been purchased during the development of the original Baldur’s Gate and was showing its age. This Project Manager System would have a data driven work-flow engine that tracks game features and their state (including any bugs associated with them). The additional features on top of standard bug-tracking that were being requested simply were unavailable in “reasonably” priced third-party packages. Development started in the fall of 2000 and took five man-months of design and implementation.

Unfortunately, those additional features that were unavailable except in the more expensive packages were not very well received and went largely unused. Since then, seven additional man-months of effort have been spent on support, maintenance and general improvements but for all intents and purposes, it remains only a bug-tracker.

Considering the scope of the program and how few of its features were ultimately used to their full potential, the Project Manager System was not as successful as it could have been. In hindsight, a third-party piece of bug-tracking software should have been purchased to allow the programming resources that were spent on it to be spent directly on Neverwinter Nights. That being said, the system has been and continues to be used on several other projects although it has finally been scheduled for replacement around mid-2004 (by a third-party product).

5 Timing

Given enough tools programmers who know their roles and responsibilities relative to the project, how can they be used most effectively? This section is broken up into the various phases of production that most projects experience.

5.1 Pre-Production

Most tools development should take place during pre-production. The goal of the pre-production phase, from the tools perspective, is to create a stable, reliable and efficient set of software for the purposes of creating the game. That is, ideally, at the end of pre-production, it should be possible to get representative samples of all the different kinds of content into the game. Obviously, it does not always work out that way.

Bring a senior tools programmer onto the project at the same time as the other leads. Together, they can determine which tools can be brought forward from previous projects unmodified or with modification (if any) and which tools must be implemented from scratch. Input from the other leads will help the project’s senior tools programmer determine the number of resources required to achieve the schedule. This should provide a sanity check on the project’s goals and schedule if those resources are unavailable and initiate the hiring process (see above) if necessary.

Once the schedule and planning have been done, bring the necessary tools programmers onto the project. This may be a large or small number depending upon the amount of new development required and the time available. However, with all the responsibilities outlined in the previous section, it is difficult to have too many tools programmers at this point in the project.

5.2 Production

Ideally, once the project enters the production phase, tools development should focus on minor modifications and enhancements (i.e. maintenance). While major systems changes may occur, they should be the exception rather than the rule. Realistically, some game systems may not be complete at the time production starts and so the corresponding tools (if any) will still have to be designed and implemented.

Gradually, over the course of production, it should be possible to reduce the number of tools programmers on the project. These programmers should be allocated to other projects, some of which may be in different phases at that time. This reflects the fact that as the project gets closer to completion, tools cost more. This is because there is less time for the development to pay for itself.

Keep in mind that once a project enters production, there should be a minimum number of tools programmers working on it up to ship-day. At BioWare, this number has recently risen from two to four. The number of systems for which the tools programmers are responsible is large (and it is growing) and their complexity is increasing. With only two people it is too easy for a single crisis to spiral out of control and it as at the end of the project when a loss of control can be fatal.

Star Wars: Knights of the Old Republic (Xbox) had only two tools programmers at the end of the project. Frequently tracking down build problems, sometimes caused by bad processing or malformed art content, kept these programmers too busy to deal with the more mundane issues of incredibly long build times. Once the Xbox version had shipped, the final push for the PC version began. At this time, additional tools resources became available and issues like build time could be addressed.

5.3 Post-Production

Post-production is the phase where patches and additional content are provided to the customer. It may also include creating a demo, press-builds and performing any localization tasks. The level of post-ship support required will largely determine the amount of tools resources necessary.

Theoretically, the project should not require more tools resources during post-production than it needed at the end of production. The amount of new development and maintenance on the tools during this phase should be extremely low. All this will depend upon things such as whether the product included an end-user toolset.

Regardless, the post-production demands for tools support should be known at least several weeks before completion. If it is not, plan for the worst until more information is available.

5.4 Cross-Project Teams

Localization and post-ship support are complex tasks and require months of work both before and after the game ships. This has prompted BioWare to create the Localization and Live teams staffed with dedicated and specialized resources, including tools programmers. While this work is just as susceptible to the pressures of delivering product to the retail market, the amount of actual new, game-related development is quite low. As such, the staffing requirements for these teams at BioWare are different than that of a full project – typically one to three programmers.

6 Success and Growth

6.1 The Ever-Rising Bar

Successfully completing one project usually leads to another. The subsequent project may be broader in scope, more technically challenging or on a shorter schedule. Regardless of the specific details of the new project, the demands on and expectations of the tools programmers will increase.

The Neverwinter Nights Toolset, for example, a cornerstone of that game, required close to 10 man-years of effort, not including the game and graphics engine systems it uses. This was in excess of what was spent on all the tools for the entire Baldur’s Gate series between 1996 and 2001. With Neverwinter Nights, a precedent was set for tools on Star Wars: Knights of the Old Republic. Fortunately, it was possible to re-use much of what was done on Neverwinter Nights for Star Wars: Knights of the Old Republic, but it still required four additional man-years of work and consequently set the expectations for Jade Empire even higher. This trend is expected to continue.

A rising bar is good thing. It helps drive and challenge the team, an essential part of keeping the team together (discussed above). In providing better tools, a rising bar also helps the project. The key is to expect that bar to continue to rise, then plan for it, hire for it and train for it.

6.2 Effects of Departmental Change

As BioWare continues to hire more technically-minded artists, designers, producers and QA analysts, the demands placed on the tools team change. Now, the customers are able to satisfy some of their own tools-related needs, needs that do not directly affect the game content. This has allowed the tools programmers to focus on the areas of stability, reliability and efficiency for the asset management, build construction, content creation, distribution, end-user and localization tools. For example, some technical designers familiar with SQL are able to access the database server to write their own reports. Technical artists work to streamline content creation from within their 3D modelling package.

This is where the greatest advantage of a dedicated tools programming team lies, ensuring the stability and reliability of the game construction processes and tools.

7 Summary

A dedicated tools programming team can offer a lot to a company. Enough of the right programmers who understand their roles and responsibilities added to the project at the right time can make a significant difference in its final quality. Such a team is not easy to build; it takes time to find the right people and a willingness to offer those people reasons to stay for the long term. Once there, keep the tools programmers focused on the helping the rest of the project team get it done faster and to the highest possible quality.

Appendix 1: Operations and Change Management

The following is the guideline for operations and change management process now followed by tools programmers at BioWare. As a guideline, it represents the ideal case and serves as a sanity check that the tools programmers can use to compare their actions against but which is “set in jello” or firm but flexible. This gives the tools programmers the freedom they need to adapt to the short-term needs of the project while still being aware of their long-term responsibilities. It can also be used to explain how things are done to any new staff on the project, tools programmers or otherwise.

By default, tools development proceeds on a five-day cycle (usually Monday – Friday). Five day cycles offer a reasonable compromise between the need for the frequent releases required to address minor issues and enough time to add major features. This is not to say that some tasks will not take more than five days to finish, but features that large can be spread over multiple cycles and released iteratively.

Day 1: New software is released from the stage into the production environment, followed by a review of the roll-out and planning with the customers (art, design, programming, production, QA, etc.). Day 1 should not be a Friday.

Day 2 – Day 4: Development. See Appendix 2. During development, anyone may receive a sneak build of a tool at any point for verification of features or fixes as necessary but which will not be used to affect production data.

Day 5: Code reviews and integration testing. This may involve volunteers from one or more customer groups. If all integration tests pass and the code reviews do not reveal any fundamental flaws, the source for the candidate application (including any game or graphics code it uses) is labelled in source control and the candidate application is staged for release at the start of the next cycle.

Changes can be made to the week’s plan as the needs of the project require with the understanding that, in general, something on the plan that hasn’t been completed will be delayed. Changes like this typically do not affect the release strategy.

Emergencies are a fact of life. Sometimes things happen over which one has no control and any process (guideline or not) that does not account for that possibility will simply be abandoned. For the purposes of this process, an emergency is defined as a data-corrupting, crash-inducing or otherwise blocking bug in one of the tools. Note that there is no such thing as an emergency feature request.

If an emergency occurs mid-week, a branch is taken from the previously approved build and the minimum work necessary to remove the block as quickly as possible is performed. The corrected software is then released and the branch labelled. This ‘fix’ may be as simple as just disabling the button that activates the problem feature in order to prevent users from accidentally triggering it or it may be the correct solution in which case it has to be merged back with the mainline in time for the next official release.

The roll-backs, branches and re-labels are necessary to account for the possibility that multiple emergencies may occur in the same week. In that case, you do not want to accidentally re-introduce a bug removed by a previous emergency release. Emergency fixes can be released quickly, but they require a lot of tools programmer time that would otherwise be spent on development. This extra overhead should help further reduce the occurrence of requests for relatively minor requests as emergencies.

Appendix 2: Development

The following is the guideline for development now followed by tools programmers at BioWare. As a guideline, it represents the ideal case and serves as a sanity check that the tools programmers can use to compare their actions against but which is “set in jello” or firm, but flexible. This gives the tools programmers the freedom they need to adapt to the short-term needs of the project while still being aware of their long-term responsibilities. It can also be used to explain how things are done to any new staff on the project, tools programmers or otherwise.

The process involves three basic steps: Design, Implementation and Shakedown. Each step includes documentation and review stages that could result in another iteration of that particular step.

Design: This is pretty basic. Talk with the customer, design the system based on those discussions, document the design and review the document with some one else. Then, either repeat or proceed based on the feedback from the review. Use this time to determine environmental conditions such as data-load and user count.

Implementation: This, too, is pretty basic although it does borrow a little from various methodologies that talk about test-first or test-driven development. Create the stubs required for the tests, write the test cases themselves and implement the stubs while continuing to run the tests to ensure they succeed. Repeat this step until all the stubs have been implemented and all of the tests work.

Shakedown: This is the point at which the primary customer contact for the tool does the final review of the program before giving the “go” to put the program into general production. Theoretically, most of the technical issues should have been worked out during design and implementation, but that is not always the case. It also allows the customer to get more of a feel for how the real program will work instead of having to rely on any prototypes that may have been built.

Note that there is a lot of explicit and implicit documentation in this process. Accept the fact that in game tools documentation is essential because of their presence on the critical path of getting data into the game and the potential they have to go for long periods (i.e. multiple projects) between changes. Understand that the only thing worse than no documentation is inaccurate documentation and be prepared to allow the tools programmers to spend the time necessary to maintain the documentation they create.

At BioWare, the emphasis is on making tools documentation useful for the author during design and implementation of the tools themselves not just during maintenance. This immediately raises the buy-in from the programmers because they can see that it’s not just make-work. This is achieved by focusing on the interfaces the software will present to its environment both internally and externally while avoiding, as much as possible, specific implementation details which are usually subject to change.

Appendix 3: Good Books

The following are a few good books that apply, to varying degrees, to the kind of work for which tools programmers are responsible.

“Failure Is Not an Option: Mission Control from Mercury to Apollo 13 and Beyond” by Gene Kranz

“Flight: My Life in Mission Control” by Chris Kraft

“Documenting Software Architectures: Views and Beyond” by Clements et al

“Software Configuration Management Patterns” by Stephen Berczuk

“Extreme Programming Explained” by Kent Beck