From the Desk of Paul Higginbottom

I have spoken on many occasions to developers about the way they develop products, and I am amazed at how little planning some developers do. It seems that many start a major project without ever creating some sort of written specification or at least a prioritized list of features. Some developers do not consider the needs of their potential customers when designing their product. Other developers do not consider the necessity of taking the time to properly test and debug a product.

This issue I'd like to talk about the development process and suggest some guidelines to follow when planning and developing your products. I believe that without proper planning, your product is likely to be buggy and late, and, if it is not a complete failure in the market, it will never approach the level of success it could have.

Step 1. Write a specification

> You may be asking ‘‘Why should I?’’ Writing a specification that actually ends up on paper and is circulated to everyone involved in the project will make sure that certain questions are answered *before* the coding begins--a question like ‘‘What kind of person is likely to use this product?’’ If your product is a productivity tool, your specification should answer how the product will make its user more productive or how it will improve the quality of the user's work. If your product is recreational, your specification should identify what is compelling about your product, so your development will capitalize on what makes it different and interesting (not ‘‘just another shoot ’em up’’).
>
> Writing a specification will give you the opportunity to identify which features are essential to the product, which would be useful but are not essential, and which might be good to add if there's time. By identifying and prioritizing the features in your product, you can more accurately predict how long the product will take to complete, and, therefore, plan your cash flow, resources needed, etc.
>
> Writing a specification is time consuming, but it can and should be fun. Everyone involved in the project should get together and brainstorm. You may even involve people who are not in the slightest bit related to the project, just to get a different perspective on things.
>
> Once you have a draft specification, you may want to refine it so that the project will evolve over several release versions. This will allow you to implement all of the features you want your product to have while giving you a reasonable time frame to implement the features properly. To help the marketing folks and to track your own progress, you should do a realistic schedule to go with the spec.

Step 2. Use the Style Guide (or CDTV UI Guidelines)

> I've certainly heard developers ask ‘‘Why should I conform to UI standards?’’ By providing the user with a familiar environment (familiar from having previously used the system or other applications), the user will have less to learn when trying to operate your program, which will lower the amount of explanation necessary in the documentation. This will also lower the number of support questions you get on the phone and will improve interoperability with other software, making your product more useful. Ultimately, *standards will save you money*.

Step 3. Use the debugging tools (*Enforcer*, *Mungwall*, etc.)

''I don't need debugging tools--I write perfect code.'' Yes, I've heard such comments before. Our debugging tools can find bugs that otherwise would have gone undetected by you. They will humble the most talented programmers into admitting that a bug exists. They can find uninitialized pointers, areas of memory you've stepped on without meaning to, areas of memory your program is using after it's given it back to the system, and much, much more. And these tools are so simple to use, it's silly not to use them. The result of using them will be much more reliable, solid products.

Step 4. Don't program empirically---follow the Amiga programming guidelines

Just because a feature works doesn't mean that it has been implemented properly. I've seen products where features or algorithms work because various parameters in the code were ''played with'' until it ''appeared'' to work properly. Take the time to find out *why* it works. Maybe it's just pure luck that certain incorrect parameters or procedures work. You may not be so lucky in the future. Future operating system releases may not be tolerant of these incorrect values or procedures because they shouldn't have worked in the first place.

Step 5. Make testing a way of life, not a last-minute exercise.

This will lower your stress level by not having a last-minute bug-fixing marathon right before you're supposed to ship your product. It will also improve the quality of your product overall, because if you only do last-minute testing, you're guaranteed to ship a product with bugs, some of which might be introduced as a result of last minute ''fixes.'' This can hurt your company's reputation, and directly affect your success. The customer's initial reaction to your product will strongly affect their future decisions, what they recommend to others, etc. Make sure your product doesn't crash!

As a normal part of your development process, let people not normally associated with your company test your product before you release it. We in CATS really like seeing new programs, and are happy to test things for you. We may not always be able to provide a detailed quality-assurance type of report, but we can give you feedback as users.

Step 6. If appropriate, include our hard disk installation utility.

If the user is likely to install your product on a hard disk, include our hard disk installation utility. The installer's online help and GUI make it much more friendly than an AmigaDOS script. It's smart enough to handle a wide variety of installation needs, yet relatively easy to configure. We've saved you the time involved in writing your own!

In summary, following these guidelines will help ensure that your products will be of the highest quality, and will help eliminate a lot of the headaches involved in the development process.

As I mentioned in my last column, I like talking to developers, so if you have any questions, suggestions, or criticism, feel free to give me a call. My number is (215) 431-9228. You can send email to me care of someone in CATS on BIX if you wish, or through uucp mail to ''higgin@cbmvax.commodore.com'' or ''{rutgers| uunet}!cbmvax!higgin''.

-PH