12 Things to Ask Your Software Team

By John V. Levy, John Levy Consulting LLC, and Geoff Caras, The Igneous Group
SEPTEMBER 22, 2003

Below are 12 questions that you should ask the people who are responsible for developing software for your company. Along with helping the development team's organization and preparation, these questions may help higher-level executives who aren't familiar with software development learn a few things about the process.

As you listen to the answers, watch for signs that your questions were unexpected or considered unreasonable or challenging by the managers who own the project. If you do see these signs, more communication is required to ensure that everyone knows that these questions don't concern arbitrary demands, but are fundamental to the business and in line with company strategy.

1. Who are the intended users of the software?

Without knowing who is going to use the software, the development team has no criteria for dealing with user interactions. Are they consumers, technicians, grocery store bookkeepers, managers? How technically proficient are they? How much time will they spend interacting with the software each day/week/month? Have they used systems like this before? Will this system be a disruptive factor in their current jobs? Do they have to understand the error messages or correct the database contents?

If the development team doesn't know the answers to these questions, go back to your product definition phase and do some more work on defining the user.

2. What are the functional objectives of the project?

There must be a few key objectives -- functions that the software will perform -- that are foremost in the developers' minds during the project. There may be a long list of features, functions and capabilities, but they should be prioritized so that the team can concentrate on the key ones. If the leaders of the project can't list the top three functions without reference to paper, then they aren't in tune with what the project is all about.

3. What is the architecture of the software you are building?

Similarly, the project leaders should be able to draw a diagram of the top-level structure of the software. This is a map of how the major modules fit together. Each major module should have a function that is clear and related to the overall functional objectives of the project. Connections between modules represent communication -- places where signals (information about what to do or when to do it) and data structures (packaged information) are exchanged.

There shouldn't be a rat's nest of lines connecting these major modules, nor should there be simply a page of unconnected boxes. Check to see whether the team understands the architecture, or at least the function of the component they are working on.

4. Which modules need to change if there is a change of processor?

Below the top-level architecture, there should be a diagram of all modules in the system. When the software system is moved into a different computing environment from the one that's initially envisioned, some of these modules will have to change. Why should you care? Because change to underlying hardware happens regularly for long-lived software. And you expect this software to last for a while, right?

5. Which modules have to change when you add new features?

Being able to explain which modules are involved in the implementation of each feature is a good indicator of familiarity with the programs. Also, if you find that most suggested additional features are going to cause change to nearly all of the modules, then you have an architecture that isn't very modular. Modular design contains the processor, system and data-specific functions in relatively few places. It's a good thing to encourage.

For example, be sure that the designers have put processor-dependent things like byte ordering, numeric precision and pointer math in logically isolated modules in order to make these fundamental definitions easier to change when the processor is changed.

Check on the feature request process. Is there a working document for feature development that makes it clear how a feature change in one area of the system may affect other areas? Is the document up to date?

6. What language are you using to implement the software?

You don't need to know a lot about computer languages to know that there are relatively few well-supported ones to choose from. Possible languages include C, C++, Java, Visual Basic and SQL. This is basic information, of course, but there are important issues related to the choice of language.

Ask the name of the vendor that supplies the language compiler and debugging tools. Ask how long it's been in business and how many users it currently supports. Ask the team how long they have been using these tools from this vendor and how good the support has been. Remember, you're going to be depending on the tools for as long as you are supporting the customers of this software.

Check to see if there is a system architect involved in the decision process. Are there adequate tools to make this project manageable? Have they checked references from the tool vendors?

7. How large do you expect the programs to be?

There are several reasons for estimating code size. It helps you to anticipate the effort involved in generating the code. It also gives a first estimate of the size of the software package you will be shipping or using, and that may affect the choice of medium you use for distribution. Finally, size corresponds roughly with complexity, so the larger the programs, the more interactions and features there will be to test and debug.

Don't automatically ask the programming staff to minimize code size, because that could cause them to spend too much time on that single issue. On the other hand, if they have no idea how big the code will be, they probably don't yet understand the requirements and specifications, or they haven't chosen a language for implementation yet.

The best approach is to make it work first, and then make it better. Trying to optimize during detailed development and design leads to designs that are "too smart" for their own good.

There are many things that can be optimized, such as debugability, memory space and fast response time. If someone is asking for one of these optimizations, be sure there is someone on the team who knows how to do it.

8. How will you know when the software is ready for release?

This is not the question, "Is the software ready for release?" This is a

ADVICE Column

Geoff Caras

John Levy

Geoff Caras is co-founder and CEO

question to your staff asking what the criteria are for determining that the software is ready for release. It is extremely important that you get the project managers to state these criteria clearly and in advance, because when the pressure comes to ship the product, it will be tempting to ignore some or all of the criteria. Make sure that some of the criteria include actual use of the product by people who aren't part of the development team, and resolve all of the questions and problems that came up during that use.

This should have been clearly outlined in the specification stage. Who makes this determination? How will you be able to tell if the functional objectives are met? What are the metrics used to evaluate/measure these objectives? Who handles this and how often?

of The Igneous Group Inc., an Internet technology and consulting service company. He has over 23 years of experience in technology management and speaks at technology and business events, workshops and forums. He has a degree in earth sciences from the University of California, Santa Cruz.

John V. Levy is principal of John Levy Consulting LLC. He has a Ph.D. in computer science from Stanford University and an M.S. in electrical engineering from the California Institute of Technology. He is an industry speaker and author specializing in software development management.

9. When can we try out the user interface?

If the user interface isn't available for testing well before the completion of the rest of the functions of the program, there will be too little time to work on redesign of the parts that aren't easy to use. Be sure that the project schedule shows user interface testing -- by users outside the development team -- earlier than halfway through the proposed schedule. Better yet, include demonstration of the user interface as part of the first review of the product specifications.

The highest priority should be the parts of the product development, such as user interface, that are most likely to need rethinking. This will help to ensure that the team will have enough time in the rest of the development cycle.

10. Which features can we try out tomorrow?

This question is aimed at finding out what's actually implemented and debugged in the software package. Often a status report will say that x, y and z are completed, but you can't use those items because another portion of the package is still incomplete. Try insisting that the project reporting include only demonstrable working features in the "completed" column. Then you can probe further by asking what tests those features have and have not passed.

Make sure that your team builds "stubs" -- modules that do nothing now, but are place-holders for future functionality -- so the whole system can be put together frequently for testing.

11. When can we try out the final feature set?

Project teams will often leave implementation of the most difficult features to the last minute. Therefore, the project isn't 50% complete when 50% of the features have been implemented. In general, you can expect that the project is about 50% complete when all of the features have been implemented. The rest of the project time will be taken up by debugging those features and the interactions between program modules. Therefore, the answer to this question may give you an indication of when the project will be approaching the halfway mark.

12. What remains to be done?

The project team will continually discover additional modules, subfunctions and rework that need to be taken care of as they develop the product. This is natural. Therefore, each time that you ask, "What have you completed?" you'll get a list of things that are done, but you won't find out what's been added to the list since the last report. So ask this question, instead, and make your own projection of completion based on the historical rate of increase in things to be done.

Be sure that your managers are using project management tools on a regular basis -- updating the information weekly or more frequently. Check that they are using the tools to let team members know what has to be done by when.

Finally, as you ask these 12 questions and deal with the resulting issues and concerns, remember that software development teams function better when they are exposed to the business issues associated with a project and that there are nontechnical issues that are affected by the schedule, budget and other factors. Keeping everyone on the same page helps to ensure a successful project.

Source: Computerworld