|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 1   |  |  | | --- | --- | | 1: | In the Year 2000 bug example, did Dave do anything wrong? | | A1: | If Dave was a good programmer he would have questioned the "obvious" oversight and not just programmed his software to only work through 1999. Because he didn't, a software tester should have tested for and found the bug. The team could have then decided whether to fix it. | | 2: | True or False: It's important what term your company or team calls a problem in its software. | | A2: | False. It's not important, but the term used often reflects the personality of the team and how they approach the finding, reporting, and fixing of the problems. | | 3: | What's wrong with just testing that a program works as expected? | | A3: | At most, that's only half the testing problem. Users don't always follow the rules, and testers need to prove out what happens when they don't. Also, if testers don't approach their testing with a gotta-break-it attitude, they will miss bugs. | | 4: | How much more does it cost to fix a bug found after the product is released than it does from the very start of the project? | | A4: | From 10 to 100 times or even higher! | | 5: | What's the goal of a software tester? | | A5: | The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed. | | 6: | True or False: A good tester relentlessly strives for perfection. | | A6: | False. A good tester knows when perfection isn't attainable and when "good enough" is reached. | | 7: | Give several reasons why the product specification is usually the largest source of bugs in a software product. | | A7: | Often a spec isn't even writtenremember, if you can't say it, you can't do it. Other reasons are that the spec exists but it isn't thorough, it's constantly changing, or it's not communicated to the rest of the development team. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 2   |  |  | | --- | --- | | 1: | Name several tasks that should be performed before a programmer starts writing the first line of code. | | A1: | The development team needs to understand the customer requirements and define the features in a product spec. A schedule should be created so team members know what work has been completed and what work remains to be done. The software should be architected and designed, and the test team should start planning their work. | | 2: | What disadvantage is there to having a formal, locked-down specification? | | A2: | There's no flexibility to adjust the software if the customer's needs change or the market moves in a different direction while the software is being developed. | | 3: | What is the best feature of the big-bang model of software development? | | A3: | It's simple. Period. | | 4: | When using the code-and-fix model, how do you know when the software is ready to release? | | A4: | There's no real exit criteria for the code-and-fix process except for when someone, or the schedule, says that it's time to stop. | | 5: | Why can the waterfall method be difficult to use? | | A5: | Just like with salmon, it's difficult to swim upstream. Each step is a discrete, standalone process that follows the one before it. If you get to the end and find that something should have happened further up, it's too late to go back. | | 6: | Why would a software tester like the spiral model better than the others? | | A6: | They're involved very early in the development process and have the opportunity to find problems early, saving the project time and money. | |
| Chapter 3   |  |  | | --- | --- | | 1: | Given that it's impossible to test a program completely, what information do you think should be considered when deciding whether it's time to stop testing? | | A1: | There is no universal answer for when to stop testing. Each project is different. Examples of the information that might go into the decision would be: Are lots of bugs still being found? Is the team satisfied with the number and types of tests that have been run? Have the reported bugs been evaluated to decide which ones to fix and which ones not to fix? Has the product been validated against the user's requirements? | | 2: | Start the Windows Calculator. Type 5,000-5= (the comma is important). Look at the result. Is this a bug? Why or why not? | | A2: | The answer you get is 0, not 4095 as you would expect. The reason is that the , (comma) was automatically converted to a . (decimal point). What you calculated was 5.000-5=0, not 5,000-5=4995. To determine if this is a bug, verify this operation against the product specification. It might state that commas are to be converted to decimal points. You should also validate this "feature" against the user requirements. Find out if most users want this to occur or if it would be confusing. | | 3: | If you were testing a simulation game such as a flight simulator or a city simulator, what do you think would be more important to testits accuracy or its precision? | | A3: | The purpose of a simulation game is to put the player into an artificial environment that mimics a real-world scenario. Flying a flight simulator should look and feel like flying a real plane. Running a city simulator should reflect what happens in a real city. What's most important is how accurately the simulator reflects reality. Does the plane fly like a Boeing 757 or Piper Cub? Does the skyline of the city look like the city it represents? Once you know the software is accurate, you can then test for precision. This is where you start caring about whether the windows in the buildings are in their precise locations and whether the plane movement perfectly maps to the joystick motion. | | 4: | Is it possible to have a high-quality and low-reliability product? What might an example be? | | A4: | Yes, but it depends on the customer's expectations for quality. Many people buy high-performance sports cars that are considered to be high quality for their acceleration and speed, their style, and their fit and finish. Often these same cars are notoriously unreliable, frequently breaking down and being expensive to repair. The owners have their own perception of quality and don't see low-reliability as an important issue. | | 5: | Why is it impossible to test a program completely? | | A5: | With any software other than the smallest and simplest program, there are too many inputs, too many outputs, and too many path combinations to fully test. Also, software specs can be subjective and be interpreted in different ways. | | 6: | If you were testing a feature of your software on Monday and finding a new bug every hour, at what rate would you expect to find bugs on Tuesday? | | A6: | Two axioms come into play here. The firstthat the number of bugs remaining is proportional to the number of bugs you've already foundmeans that you won't come in on Tuesday and miraculously find the software to be perfect. The pesticide paradox tells you that if you continue to run the same tests over and over that you eventually won't find new and different bugs until you add more tests. Given these two axioms, you will probably continue to find bugs at the same rate or slightly less. | |
| Chapter 4   |  |  | | --- | --- | | 1: | Can a software tester perform white-box testing on a specification? | | A1: | Yes, white-box testing is simply using information about how something is designed to influence how the testing is done. The tester could attend the project's focus groups, usability studies, and marketing meetings to understand the underlying process being used to design the features and the overall product. With that information she could more effectively test the resulting specification. There is a risk, though, that this information could bias the tester into assuming that the spec is correct. | | 2: | Cite a few examples of Mac or Windows standards or guidelines. | | A2: | On the Mac, deleted files go to the Trash Can. In Windows, they go to the Recycle Bin.  In Windows, pressing F1 always displays Help for the software. It's Command-? on the Mac.  The File menu is always the far-left menu choice in Windows.  In Windows, selecting Help, About displays the software's copyright, licensing, and version information.  On a Mac, Command-X performs a cut, Command-C performs a copy, and Command-V performs a paste.  These are just a few examples of a wide variety of possible answers. | | 3: | Explain what's wrong with this specification statement: When the user selects the Compact Memory option, the program will compress the mailing list data as small as possible using a Huffman-sparse-matrix approach. | | A3: | It uses the phrase as small as possible. This is impossible to test because it's vague and imprecise. The spec should state exactly what level of compression will take place.  The statement also isn't code-free. It explains how the feature will work on the algorithm level. This doesn't belong in a specification. The user doesn't care how the compaction occurs, just that it does. | | 4: | Explain what a tester should worry about with this line from a spec: The software will allow up to 100 million simultaneous connections, although no more than 1 million will normally be used. | | A4: | Testability. It doesn't matter that typical usage is only 1 million connections. If the specification states that 100 million are possible, the 100 million must be tested. The tester needs to find a way to test something this large or get the spec writer to reduce the maximum possible number to something closer to what's typical. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 5   |  |  | | --- | --- | | 1: | True or False: You can perform dynamic black-box testing without a product specification or requirements document. | | A1: | True. The technique is called exploratory testing, and you essentially use the software as though it's the product spec. It's not an ideal process, but can work okay in a pinch. The largest risk is that you won't know if a feature is missing. | | 2: | If you're testing a program's ability to print to a printer, what generic test-to-fail test cases might be appropriate? | | A2: | You could attempt to print with no paper and with jammed paper. You could take the printer offline. You could unplug its power. You could disconnect its cable. You could try printing with low ink or toner, maybe even with a missing cartridge. To identify all the possibilities, you could look in the printer's operator's manual and see what errors it supports and attempt to create all of them. | | 3: | Start up Windows WordPad and select Print from the File menu. You'll get the dialog shown in Figure 5.12. What boundary conditions exist for the Print Range feature shown in the lower-left corner?  Figure 5.12. The Windows Print dialog box showing the Print Range feature. | | A3: | If you select the Pages option, the From and To fields are enabled. The obvious boundary conditions would be 0 and 99999, the smallest and largest values that the fields will hold. It would be wise to also test internal boundary conditions such as 254, 255, 256 and 1023, 1024, and 1025. There's also another internal boundary condition. Try asking it to print pages 1 though 8 of a 6-page document. Notice that in this case, the software has to stop printing at page 6 because it's out of data, not because it was told to stop at this page. It's a different, internal boundary. See if you can think of more. | | 4: | Assume that you have a 10-character-wide ZIP code text box, such as the one shown in Figure 5.13. What equivalence partitions would you create for this text box?  Figure 5.13. A sample ZIP code text box that holds up to 10 characters. | | A4: | You should have at least these four equivalence partitions, although you may think of more:  Valid 5-digit ZIP codes. Valid means that they're numeric digits, not that they are existing, in-use ZIP codesalthough that could be another partition.  Valid 9-digit (9 digits with a dash) ZIP codes.  Short 5-digit. Have only 4 numbers, for example.  Short 9-digit. Have only 8 numbers, for example.  Long 5-digit. Have 8 digits without a dash, for example. Hmm, is this the same as a short 9-digit partition?  Long 9-digit. It may not be possible to type in more than 9 digits and a dash, but you should try.  10 digits, no dash. This is a little different than a long 9-digit partition.  The dash in the wrong place.  More than one dash.  Non-digit and non-dash entries. | | 5: | True or False: Visiting all the states that a program has assures that you've also traversed all the transitions among them. | | A5: | False. Think of visiting 50 different cities spread out across the entire United States You could plan a trip that would take you to each city, but it would be impossible for you to travel all the roads that connect to all the citiesthat would be all the roads in the entire country. | | 6: | There are many different ways to draw state transition diagrams, but there are three things that they all show. What are they? | | A6: | Each unique state that the software can be in.  The input or conditions that make it move from one state to the next.  Conditions, variables, or output that's produced when a state is entered or exited. | | 7: | What are some of the initial state variables for the Windows Calculator? | | A7: | The initial displayed value and the internal partial result are set to 0. The memory register (MC, MR, MS, and M+ buttons) is set to 0. The Clipboard contents (where data goes when you cut, copy, and paste) is left unchanged.  Another initial state variable is where the Calculator appears onscreen when it's started. Open up several copies of the Calculator and notice that the location isn't always the same (at least on Windows 95/98). As an exercise in exploratory testing, see if you can figure out what the rules are for where the Calculator opens. | | 8: | What actions do you perform on software when attempting to expose race condition bugs? | | A8: | You try doing multiple things at the same time. They could be related, like printing to two printers at the same time from the same application. Or, they could be unrelated, like pressing keys while a computation is taking place. What you want to do is force a situation where the software is competing (racing) with itself to perform some function. | | 9: | True or False: It's an unfair test to perform stress testing at the same time you perform load testing. | | A9: | False. No test is ever unfair. Your job is to find bugs. But, the realities of software testing may apply and any bugs you find in this situation may not be fixed. | |
| Chapter 6   |  |  | | --- | --- | | 1: | Name several advantages to performing static white-box testing. | | A1: | Static white-box testing finds bugs early in the development cycle, making them less time-consuming and less costly to fix. The software testers can gain information about how the software works, what potential weaknesses and risky areas exist, and can build a better working relationship with the programmers. Project status can be communicated to all team members who participate in the testing. | | 2: | True or False: Static white-box testing can find missing items as well as problems. | | A2: | True. Missing items are arguably more important than normal problems and can be found through static white-box testing. When the code is checked against published standards and guidelines and carefully analyzed in formal reviews, missing items become obvious. | | 3: | What key element makes formal reviews work? | | A3: | Process. Having a process that's followed is what makes the difference between a formal review and two pal programmers glancing over each other's code. | | 4: | Besides being more formal, what's the big difference between inspections and other types of reviews? | | A4: | The key difference is that with inspections, a person other than the original author of the code is the presenter. This obliges another person to fully understand the software being inspected. It's much more effective than having others simply review the software for bugs. | | 5: | If a programmer was told that he could name his variables with only eight characters and the first character had to be capitalized, would that be a standard or a guideline? | | A5: | That would be a standard. If he was told that a name of more than eight characters was preferred, that would be a guideline. | | 6: | Should you adopt the code review checklist from this chapter as your team's standard to verify its code? | | A6: | No! It's provided as a generic example only. There are some good test cases in it that you should consider when you test your code, but you should research and read about other published standards before adopting your own. | | 7: | A common security issue known as a buffer overrun is in a class of errors known as what? They are caused by what? | | A7: | Data Reference. They are caused by using variables, constants, arrays, strings, or records that aren't properly declared or initialized for the data that they can contain. | |
| Chapter 7   |  |  | | --- | --- | | 1: | Why does knowing how the software works influence how and what you should test? | | A1: | If you test only with a black-box view of the software, you won't know if your test cases adequately cover all the parts of the software nor if some of the cases are redundant. An experienced black-box tester can design a fairly efficient suite of tests for a program but he can't know how good that suite is without some white-box knowledge. | | 2: | What's the difference between dynamic white-box testing and debugging? | | A2: | Both processes overlap, but the goal of dynamic white-box testing is to find bugs and the goal of debugging is to fix them. The overlap occurs in the area of isolating exactly where and why the bug occurs. | | 3: | What are two reasons that testing in a big-bang software development model is nearly impossible? How can these be addressed? | | A3: | With the software delivered to you in one big piece, it's difficult, if not impossible, to figure out why a bug occursthe needle-in-a-haystack problem. The second reason is that there are so many bugs, some hide others. You'll grab one and shout "gotcha" only to discover that the software is still failing.  Methodically integrating and testing the modules as they are built allows you to find and fix bugs before they start to become hidden or pile on top of each other. | | 4: | True or False: If your product development is in a hurry, you can skip module testing and proceed directly to integration testing. | | A4: | True. You can, but you shouldn't. Doing so will cause you to miss bugs that could have been found earlier. Skipping or postponing testing generally makes your project take longer and cost more to complete. | | 5: | What's the difference between a test stub and a test driver? | | A5: | A test stub is used in top-down testing. It stubs out, or substitutes itself, for a low-level module. It looks and acts to the higher-level code being tested just like the original low-level module.  A test driver is the opposite of a stub and is used in bottom-up testing. It's test code that replaces higher-level software to more efficiently exercise a low-level module. | | 6: | True or False: Always design your black-box test cases first. | | A6: | True. Design your test cases based on what you believe the software is supposed to do. Then use white-box techniques to check them and make them most efficient. | | 7: | Of the three code coverage measures described, which one is the best? Why? | | A7: | Condition coverage is the best because it also incorporates branch coverage and statement coverage. It tests that the conditions within decision logic, such as if-then statements, are verified, as well as branches from those statements and the lines of code. | | 8: | What's the biggest problem of white-box testing, either static or dynamic? | | A8: | You can easily become biased. You might look at the code and say, "Oh, I see, I don't need to test that case, the code handles it properly." In reality, you were blinded by the light and eliminated necessary test cases. Be careful! | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 11   |  |  | | --- | --- | | 1: | True or False: All software has a user interface and therefore must be tested for usability. | | A1: | True. Eventually, even the most deeply embedded software is exposed, in some way, to a user. Keep in mind that the UI may be as simple as a switch and a light bulb or as complex as a flight simulator. Even if the software is a single module in a code library, its interface, in the form of variables and parameters, is exposed to a programmerwho can be a user, too. | | 2: | Is user interface design a science or an art? | | A2: | It's a little bit of both. Many user interface designs have been thoroughly tested in the labs, been through rigorous studies, only to be complete failures in the marketplace. | | 3: | If there's no definitive right or wrong user interface, how can it be tested? | | A3: | Software testers should check that it meets seven important criteria: That it follows standards and guidelines, that it's intuitive, consistent, flexible, comfortable, correct, and useful. | | 4: | List some examples of poorly designed or inconsistent UIs in products you're familiar with. | | A4: | This answer will vary based on the products you use but think about these: Try setting the time on your car radio's clockcan you do it without using the manual?  Most Windows dialog boxes have the OK button on the left and the Cancel button on the right, whereas others have Cancel on the left and OK on the right. If you get used to one layout and click without looking, you could lose your work!  Did you ever accidentally hang up on someone when you clicked the receiver hook on your phone to use call waiting or conference calling?  And, the best one of all time…is up to you to find! | | 5: | What four types of disabilities could affect software usability? | | A5: | Visual, hearing, motion, and cognitive impairments. | | 6: | If you're testing software that will be accessibility enabled, what areas do you need to pay close attention to? | | A6: | Areas dealing with the keyboard, mouse, sound, and display. If the software was written to a popular platform that supports accessibility, the test effort will be a bit easier than if the accessibility features were programmed entirely from scratch. | |
| Chapter 12   |  |  | | --- | --- | | 1: | Start up Windows Paint (see Figure 12.4) and look for several examples of documentation that should be tested. What did you find?  Figure 12.4. What examples of documentation can you find in Windows Paint? | | A1: | Here are a few examples: There's rollover helpthe little pop-up descriptions you get when you hold the cursor over a painting tool. Selecting About Paint from the Help menu displays a window with copyright and licensing information. Pressing F1 starts the online help system where you can read the manual, select from an index, or type in words to search for. There's also function helpfor example, if you select Edit Colors from the Colors menu, click the ? in the title bar, and then click one of the colors, you'll get help about choosing and creating colors. | | 2: | The Windows Paint Help Index contains more than 200 terms from airbrush tool to zooming in or out. Would you test that each of these takes you to the correct help topics? What if there were 10,000 indexed terms? | | A2: | Every testing task is a risk-based problem. If you have time to test all the index entries, you might choose to do so. If you can't test them all, you'll have to create equivalence partitions of the ones you think are important to check. You could base your decision on information you get from the programmers on how the index system works. You might talk with the writer to find out how the entries were generated. You might try one of each starting letter, or the 1st, 2nd, 4th, 8th, 16th, … and last. You could even wait until you read Chapter 15, "Automated Testing and Test Tools." | | 3: | True or False: Testing error messages falls under documentation testing. | | A3: | True. But, it's not just documentation testing. The content of the message needs to be tested as documentation, but forcing the message to appear and assuring that the correct message is displayed is testing the code. | | 4: | In what three ways does good documentation contribute to the product's overall quality? | | A4: | Improved usability, improved reliability, and lower support costs. | |
| Chapter 15   |  |  | | --- | --- | | 1: | Name a few benefits of using software test tools and automation. | | A1: | They can speed up the amount of time it takes to run your test cases. They can make you more efficient by giving you more time for test planning and test case development. They're precise and relentless. | | 2: | What are a few drawbacks or cautions to consider when deciding to use software test tools and automation? | | A2: | Because software can change during the product's development, your test tools will need to change, too. You can fall into a trap of spending too much time designing tools and automation, neglecting actual testing. It's easy to rely on automation too much. There's no substitute for testing the software yourself. | | 3: | What's the difference between a tool and automation? | | A3: | A test tool will help you test, making it easier for you to perform a manual testing task. Automation is also a tool, but it will run without your intervention. Think power saw and hammer building a house while the carpenter sleeps. | | 4: | How are viewer tools and injector tools similar and different? | | A4: | Both types of tools hook into the software at points not normally accessible to the average user. Viewer tools are non-invasive as they allow you to see only what's happening. Injector tools are invasivethey allow you not only to see what's happening, but also to manipulate it. You can try test cases that might otherwise be difficult or impossible to perform at the normal user level. | | 5: | True or False: An invasive tool is the best type because it operates closest to the software being tested. | | A5: | False. An invasive tool may provide you with better information and control in some situations, but it has the possible downside of affecting the software and the test results. It's best to carefully evaluate each situation and select the tool that works best, with the fewest side effects. | | 6: | What's one of the simplest, but effective, types of test automation? | | A6: | Recording and replaying your test cases so that you only have to manually perform them once is very effective. It frees you from the monotonous repetition and can give you more time to hunt for those hard to find bugs. | | 7: | Name a few features that could be added to the test automation you described in question 6 to make it even more effective. | | A7: | Simple programming of steps rather than captured steps. The ability to pause or wait for the software to react to the actions. Some type of simple verification so that the macros know whether a bug has occurred. | | 8: | What advantages do smart monkeys have over macros and dumb monkeys? | | A8: | They're almost self-aware. They know the software's state table so they know where they are and what they can do. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 16   |  |  | | --- | --- | | 1: | Describe the pesticide paradox and how bringing in new people to look at the software helps solve it. | | A1: | The pesticide paradox (described in Chapter 3, "The Realities of Software Testing") is the situation that occurs if you continue to test software with the same tests, or the same people. Eventually, the software seems to build up an immunity to the tests because no new bugs are found. If you change the tests or bring in new testers, you'll find new bugs. The bugs were always there, it's just that the new approach made them visible. | | 2: | What are a few positives to having a beta test program for your software? | | A2: | It gets lots of additional people looking at the software. It's a good way to find configuration and compatibility problems. | | 3: | What are a few cautions to consider with a beta test program? | | A3: | A beta test is no substitute for an organized, planned, methodical test approachit's not good at general bug finding. You should know who the beta testers are in regards to their experience level, equipment, and needs to ensure that you get what you expect out of the test. | | 4: | If you're testing for a small software company, why would it be a good idea to outsource your configuration testing? | | A4: | The expense and overhead to stock and manage a configuration testing lab is very high and would likely be prohibitive for a small company or project. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chapter 17   |  |  | | --- | --- | | 1: | What's the purpose of a test plan? | | A1: | To paraphrase the IEEE 829 definition, the purpose of a test plan is to define the scope, approach, resources, and schedule of the testing activities and to identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan. In short, to tell and get agreement from the rest of the project team exactly how the heck the test team intends to test the software. | | 2: | Why is it the process of creating the plan that matters, not the plan itself? | | A2: | Because all the issues and questions defined in a test plan either impact or are influenced by other project functional groups or team members. Getting everyone to understand and agree to the contents of the plan is what matters. Privately creating a paper document and putting it on a shelf is not just a waste of time, but also jeopardizes the project. | | 3: | Why is defining the software's quality and reliability goals an important part of test planning? | | A3: | Because left to their own, everyone will have different ideas of what quality and reliability mean to them. Since they're all different, they all can't be achieved. | | 4: | What are entrance and exit criteria? | | A4: | These requirements must be met to move from one testing phase to another. A phase can't be left until its exit criteria are met. A new phase can't be entered until its entrance criteria are met. | | 5: | Name a few typical testing resources that should be considered when test planning. | | A5: | People, equipment, offices and labs, software, outsourcing companies, and miscellaneous supplies. | | 6: | True or False: A schedule should be made to meet absolute dates so that there's no question when a testing task or phase is to start and when it is to end. | | A6: | False. Because testing depends so much on other aspects of the project (for example, you can't test something until it's coded), a test schedule is best made relative to the delivery dates. | |
| Chapter 18   |  |  | | --- | --- | | 1: | What are the four reasons for test case planning? | | A1: | Organization, repeatability, tracking, and proof of testing. | | 2: | What is ad hoc testing? | | A2: | Ad hoc testing is testing without a plan. It's easy and fun but it's not organized, it's not repeatable, it can't be tracked, and when it's over, there's no proof that it was ever done. | | 3: | What's the purpose of a test design specification? | | A3: | The purpose of the test design spec is to organize and describe the testing that needs to be performed on a specific feature. It outlines the features to be tested and the approach to be used. It identifies the test cases, but doesn't specify them, and what the pass/fail criteria is. | | 4: | What is a test case specification? | | A4: | This document defines the actual input values used for testing and the expected outputs. It also lists any special environmental needs or procedure requirements and any inter-dependencies among test cases. | | 5: | Other than a traditional document, what means can you use to present your test cases? | | A5: | Tables, matrices, lists, graphical diagramswhatever means most efficiently presents the test cases to you, other testers, and other members of your product team. | | 6: | What's the purpose of a test procedure specification? | | A6: | The purpose of the test procedure spec is to identify all the steps required to perform the test cases, including how to set up, start, run, and shut down the test. It also explains what to do in case the test doesn't go as planned. | | 7: | At what level of detail should test procedures be written? | | A7: | That's a question without a specific answer. It greatly depends on who will be using the procedures. Too little detail makes the test procedures ambiguous and variable. Too much detail can bog down the test process. The level of detail should be set by the industry, the company, the project, and the test team. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Chapter 19   |  |  | | --- | --- | | 1: | Cite a few reasons that a bug might not be fixed. | | A1: | There's not enough time in the schedule, it's not a bug, it's too risky, it's not worth it, and the bug wasn't reported properly. | | 2: | What basic principles can you apply to your bug reports to give them the best chance of getting the bug fixed? | | A2: | Log them as soon as possible. Effectively describe the bug, making sure it's minimal, singular, obvious and general, and reproducible. Be nonjudgmental in your approach. Follow the report through its life cycle. | | 3: | Describe a few techniques for isolating and reproducing a bug. | | A3: | Record what you do and review it carefully. Use white-box test techniques to look for race conditions, boundary conditions, memory leaks, and other similar problems. See if the bug is state related, such as initial state or later state dependent. Consider resource dependencies and even hardware problems as the source of the bug. | | 4: | Suppose that you're running tests on the Windows Calculator and find that 1+1=2, 2+2=5, 3+3=6, 4+4=9, 5+5=10, and 6+6=13. Write a bug title and bug description that effectively describes this problem. | | A4: | Title: Adding two even numbers gives an answer that's one too much.  Description:  Test Case: Simple addition  Setup Steps: Start Version 1.0 of Calculator  Repro Steps: Try adding two even numbers such as 2+2, 4+4, and 6+6.  Expected Result: Correct answer for all calculations2+2=4, 4+4=8…  Actual Result: For two even numbers, the answer is one too high2+2=5, 4+4=9, 6+6=13, and so on. | | 5: | What severity and priority would you give to a misspelling in a company's logo on the software's start-up screen? | | A5: | Probably Severity 3 (minor problem), Priority 2 (must fix before release). | | 6: | What are the three basic states of a software bug's life cycle and the two common additional states? | | A6: | Open, Resolved, and Closed are the basic states. Review and Deferred are two possible other states. | | 7: | List a few reasons that a database bug-tracking system is so much more useful than a paper-based system. | | A7: | You can see at a glance what a bug's life cycle has beeneven if it has been complex. The current status of a bug can be instantly known. Bugs can't be lost or neglected as easily. Project statistics can be quickly obtained. | | | Chapter 20   |  |  | | --- | --- | | 1: | If you were using metrics from the bug-tracking database to measure your progress or success at testing, why would just counting the number of bugs you find per day or computing your average find rate be an insufficient measure? | | A1: | It doesn't tell the entire story. You could be testing the most complex area of the software. Your area could have been written by the most experienced programmer. It could have been written by the least experienced programmer. The code you're testing may have already been tested or may be brand new. | | 2: | Given your answer to question 1, list a few additional software metrics that could be used to measure more accurately and precisely your personal progress or success at testing. | | A2: | Average number of bugs found per day. Total bugs found so far. Ratio of fixed bugs to all bugs found. Ratio of Severity 1 or Priority 1 bugs to all bugs found. Average time from the Resolved state to the Closed state. | | 3: | What would a database query look like (any format you want) that would extract all the resolved bugs assigned to Terry for the Calc-U-Lot v3.0 project? | | A3: | Product EQUALS Calc-U-Lot AND  Version EQUALS 3.0 AND  Status EQUALS Resolved AND  Assign TO EQUALS Terry | | 4: | If the bug-find rate for a project was decreasing like the one shown in Figure 20.8 and everyone was excited that the project was getting close to releasing, what might be a couple reasons why this wouldn't be true, that the numbers were lying? | | A4: | It's possible that the software was released to testing in phases and not all the software was tested yetit might only be leveling off for the current phase. The testers might be busy regressing and closing bugs and not looking for new ones. It could have been a very warm and sunny week or the testers might be out on vacation. | | | Chapter 21   |  |  | | --- | --- | | 1: | Why are there testing costs associated with the costs of conformance? | | A1: | Because no matter how good the development process is, testing still needs to be performed one time to verify the product against the product specification and validate it against the user requirements. If no bugs are found, great, but all the costs associated with planning, developing, and executing the tests contribute to the costs of conformance. | | 2: | True or False: The test team is responsible for quality. | | A2: | False! Testing looks for bugs. Testers didn't put the bugs in the product and can't guarantee when they're done testing that no more bugs exist. | | 3: | Why would being called a QA Engineer be a difficult title to live up to? | | A3: | Because it implies that you are the one guaranteeing the product's quality. Are you ready for that responsibility? | | 4: | Why is it good for a test or quality assurance group to report independently to senior management? | | A4: | If they report to the development manager or the project manager, there's a conflict of interest between finding bugs and the creation of the software or the meeting of the schedule. | | 5: | If a company complied with the ISO 9000-3 standard for software, what CMM level do you think they would be in and why? | | A5: | They would probably be at CMM Level 3, possibly touching some of the Level 4 requirements. They aren't at Level 2 because Level 2 is just concerned with the project level. Level 3 deals with the entire organization or company. Level 4 is where statistical control starts to come into play | | | Chapter 22   |  |  | | --- | --- | | 1: | When looking for a software testing position on the Internet, what keywords should you use in your search? | | A1: | Since the job names and descriptions for software testers are variable, you should try looking for software test, software testing, quality assurance, and QA. | | 2: | Name three ways that you can become involved in testing computer software before it is released to the public. | | A2: | Beta testing and usability testing are two good methods. A third way is testing open source software and using open source testing tools. You might even get paid to find bugs! | | 3: | What's the goal of a software tester? | | A3: | The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixedtempered, of course, with all the realities of software testing that you learned in this book. Good luck and go find those bugs! | | |