Are Bugs Expensive?
A Case Study

You've graduated from the university and have been hired by Alpha Corporation, a (fictional) custom software development company. Alpha Corporation won the contract for developing a library management system for the Liberty University System. You are on the software development team. Your boss has given you the task of developing the resource scheduler.

When a patron requests a title for a specific date, the resource scheduler tries to find a copy of that title that is available for the requested date. If there are multiple copies available, the resource scheduler should schedule the copy that has been used the least.

Now assume that as you implement the resource scheduler you accidently insert a bug that causes the most commonly used copy of a title to be preferentially scheduled. The other copies of a title are scheduled only if the most commonly used copy is unavailable.

Your code compiles.

Case 1

- ## Time Required: 4 hours

- You tell your boss that you are finished and you submit your source code to the project archives.

Case 2

- ## Time Required: 5 hours

  - You build a test program that calls your module. The test program compiles. When you run the test program, you note that copies are being scheduled. You tell your boss that you are finished and you submit your source code to the project archives.

Case 3

- ## Time Required: 8 hours

- You build a test program that calls your module. The test program compiles. You run the test program and study the inputs and outputs. You verify that if a title has multiple copies, some of which are in use for a requested date, an available copy is located and scheduled. You tell your boss that you are finished and you submit your source code to the project archives.

Case 4

- ## Time Required: 16 hours
- You build a test program that calls your module. The test program compiles. You run the test program with a set of test data to verify that the least used copy is scheduled. After studying the results, you find that the most commonly used copy is preferentially scheduled. You re-examine your code, locate and remove that error. You retest and verify that the module now performs this action correctly. You tell your boss that you are finished and you submit your source code to the project archives.

Costs - 1

Which is the cheapest course of action?

Assume that your burdened salary (salary with all benefits, and employer paid costs) is $100/hour.

What is the cheapest course of action?

Resource Scheduler Cost

Case No.	Hours	Development Cost
1	4	$400
2	5	$500
3	8	$800
4	16	$1600

Costs - 2

Assume that for cases 1, 2 and 3, your bug goes undetected and still exists in the final build that is delivered to Liberty University.

After six months, the employees at the University note that some scheduled items are wearing faster than expected. After a year it becomes obvious that items are not being scheduled so as to spread the wear evenly across all available copies of a title. Liberty University submits a DR (Defect Report) and asks to be reimbursed for the items that had to be replaced early because of excessive wear.

What is now the cheapest course of action?

Costs - 3

The bug described in the DR is added to the list of other defects in the project. In a meeting with the project manager the defects are reviewed.

The scheduling defect is an obvious contract violation. A maintenance programmer is called in and given the task of verifying that the defect exists. Once located, the maintenance programmer is to report possible solutions before fixing the defect. Assume that it takes the maintenance programmer 4 days to build a test program; assemble a set of test data to exercise the bug; test and verify that the problem actually exists and is repeatable. Assume that once located, the fix is reasonably obvious and takes only 2 hours to implement.

Costs - 3 (cont'd)

The maintenance programmer's recommendations are reviewed and approval is granted. The maintenance programmer removes the bug. The bug is documented and an additional test is added to the regression suite. The documentation department reviews the bug to see if there is any impact on the written manuals. The entire regression suite is run against the new build. The new build with updated documentation is submitted to Liberty University. The university runs their own acceptance tests on a isolated test system before calling the new build their production system. From the time the DR was submitted until the fix was placed online by the university was three months and cost a total of five man-months.

What is now the cheapest course of action?

Resource Scheduler Cost - Actual

Case No	Development cost	DR Cost (programming, documentation & review)	Contract Violation Cost	Total Code Cost
1	$400	$18,400	$20,000	>$38,800
2	$500	$18,400	$20,000	>$38,900
3	$800	$18,400	$20,000	>$39,200
4	$1600	$0	$0	$1,600

Are there cheap bugs?

The cheapest bug is one that never sees the light of day.

The cost of fixing a bug increases as the point of discovery moves away from the programmer. Why?

In the previous example, it took the maintenance programmer four days and two hours to verify, locate and fix the defect. If that process had been instantaneous what would have been the effect on the schedule and cost? As the point of discovery moves away from the programmer, the actual cost of correction becomes a small part of the total cost to correct the defect.

Given that you will insert defects into your code how can you reduce the cost of those defects?

More Questions

As you develop a block of source code, you should ask yourself the following questions:

- Does this code make sense in the global context?
- Does this code make sense in the local context?

Code clarity tends to follow the second law of thermodynamics -- it degrades with time. If the code is not clear while its context is fresh and vivid, it is unlikely to improve as your familiarity with its context fades. At some point, you or someone else will have to modify your code because the requirements will change or an error needs correcting. Clarity is essential to minimize the time and resources spent implementing the change.

How do I prevent bugs?

When you discover a defect, ask yourself the following questions:

- How could I have avoided inserting this specific coding defect?
- What habits and practices can I use to eliminate this kind of defect before they occur?

Steve Maguire in *Writing Solid Code* suggests that you ask yourself the following two questions:

- How could I have *automatically* detected this bug?
- How could I have *prevented* this bug?

Steve also says that: *The easy answer to both questions would be "better testing," but that's not automatic, nor is it really preventative. Answers like "better testing" are so general that they have no muscle -- they're effectively worthless. Good answers to these questions result in specific techniques that eliminate the kind of bug you've just found.*