Learning on Steroids:

Mental Debugging

by Scott Young

Mental Debugging

In this implementation guide, I'll share a simple way you can cut down the amount of practice problems and questions you need to do in order to perform well on tests. The tactic is easy to apply, but many people fail to use it, and as a result, blunder through otherwise easy classes.

This practice is called mental debugging. The way it works is similar to how computer programmers rigorously test a computer program to identify errors and correct them. In the most basic form it's just: test, spot errors and fix the underlying mental errors that led to them being made.

Mental debugging works under the following principles:

1. Most mistakes are due to conceptual errors.
2. Mistakes of that kind should only be made once. (That is, once you spot the bug, it should be debugged immediately)

3. The purpose of practice is to debug—not to memorize solution patterns.

It's important to go over some of these principles, because many students waste dozens of hours working through practice problems, and although there method will superficially seem similar to mental debugging, because they don't rely on these principles their effort is often wasted.

Mistakes are Due to Conceptual Errors

In a testing situation, there are probably only three mistakes you can make:

1. Mistakes because of a **conceptual error**
2. Mistakes because of **forgotten trivia**
3. **"Stupid" mistakes** due to sloppiness, time pressure, etc.

Mental Debugging

If you understand the concepts perfectly, but you can't remember whether the formula is to the power of negative n or positive n, then model debugging can't help much. It's better to go back, understand the formulas and facts before moving on. Practice problems can help reinforce facts, but they are a fairly slow way of doing that if it is your only problem.

Similarly, if you understand the concepts perfectly, but make "stupid" mistakes—practice can help. But in this case it's probably even more useful to create a testing plan to make sure you don't misread questions, answer hastily or run out of time.

Model debugging works when the mistakes you're making are because you don't fully "get" the material you're studying. If you've carefully gone through a question, didn't mess up any formulas, and still got the answer wrong—this is the problem that debugging is intended to solve.

Mistakes Should Only Be Made Once

Conceptual mistakes should only ever be made once. I say this not to be harsh, but to be efficient. Making the same conceptual mistake twice serves no purpose. After you make an error, you need to find the mental bug and fix it.

If you're going through practice problems, for example, you should keep a keen eye to any conceptual mistakes. As soon as you make one, your #1 priority should be to figure out exactly why you were wrong, and become really comfortable with the answer you get. Moving onto the next one or getting nine out of ten, isn't a good excuse if you don't understand the underlying concepts.

The advantage of this ruthless approach is that you save a lot of time doing practice questions. It doesn't usually take too much

debugging to remove most, if not all, of your conceptual errors, so after only a short amount of practice, you should be able to get every question correct (minus the errors due to "stupid" mistakes or forgotten trivia).

The Purpose of Practice is Debugging --Not Memorizing Solution Patterns

This last point is the most crucial, since it is the least observed. Many students do practice problems with the unstated goal of memorizing solution patterns. That is, they do more and more problems so that they can memorize the steps of how to solve the problem.

This should never be the goal of practicing for a few reasons:

Learning on Steroids:

Mental Debugging

1. **Memorizing solution patterns leaves you vulnerable.** Creative or different questions can catch you off-guard if they don't follow your patterned response.

2. **Complex solution patterns shouldn't be remembered by rote.** For the few classes that have incredibly complicated solution patterns for particular problems, you should use holistic methods to remember those patterns, not simply doing hundreds of questions.

3. **Memorized solutions patterns gives a false sense of security.** Since you can get the questions correct, there isn't assumed to be any underlying problem. However, if you still have mental bugs, you can get plenty of questions wrong in the future, even if your patterns are memorized perfectly.

Mental Debugging

The purpose of practicing calculus questions, for example, isn't to go step-by-step through how to create a derivative. The purpose should be to debug flaws in the way you think about derivatives, which lead to eventual errors. If you fix the mental bugs, then it only takes a small amount of practice to start getting the questions right all the time.

When you switch from “practicing to memorize” to “practicing to debug” you can also save a lot of time. Suddenly, spending dozens of hours practicing the same questions over and over again doesn't make sense. Once your model has been debugged, you can stop after getting just a few questions right.

Applying Mental Debugging

Mental debugging is fairly simple, it just involves:

1. Creating your self-tests
2. Spotting conceptual errors
3. Fixing conceptual errors

First Step: Creating Self Tests

The first step in mental debugging is to create self-tests. Because your goal here isn't to memorize solution patterns, but to debug your mental model of how a particular problem works, blindly following the assignment questions isn't very efficient.

Mental Debugging

Programmers encounter these problems when testing their software. Testing software with the same input repeatedly is a waste of time. It will deliver the same result, and it won't point out any errors in the software design.

Instead, programmers build tests that try to examine all the different possibilities of input that could be given to a section of the code. Instead of throwing a bunch of identical questions, all the different possibilities are tested, so that if an error exists, it should be brought up in at least one of them.

Similarly, you can save yourself a lot of time self-testing, by not doing every single question, but by doing one of every different type of question. Whenever you make a mistake, don't just do another problem, do another problem of the same type—if you make the mistake again, you've got a mental bug that needs fixing.

Remember also that the goal with self-testing isn't to get "good enough" but to get perfection. If you discount mistakes due to forgotten trivia or sloppiness in calculation, then you should aim to have zero mistakes due to conceptual errors.

This may be impossible in practice, but doing 20 questions and getting 2 wrong due to mental bugs doesn't mean you got 90%, since in a different problem set those same two bugs could result in only getting 40%.

Step Two: Spot Conceptual Errors

Be quick to distinguish errors made from missing trivia or calculation mistakes and errors due to missing concepts. If I run through a complex finance question, for example, and incorrectly input a formula, that's a mistake—just not a conceptual one. If I do

the same question but use the wrong formula, that's probably a conceptual error.

When doing your self-testing, pick out the conceptual errors and focus on those first. In technical, problem-heavy courses, conceptual errors cause far more damage than any other kind of mistake.

Step Three: Fix the Conceptual Error

Once you have spotted a mistake, figure out why you did what you did, and most importantly why the answer differed from the correct one. If you look over the answer key and the solution seems obvious to you in hindsight, then you're probably okay. If you're confused, you need to go back and seek out an explanation.

Mental Debugging

Even “minor” mistakes can cause problems if there are too many of them. If you frequently confuse formulas or steps in a process, you may understand the problem, just not very well. This is when you can go back to the topics at hand and create new metaphors or visceralizations to remember them accurately for the future.

Good luck with this tactic and I'll see you on the other side!