Excerpted from Chapters 2 and 3 of
Debugging by Thinking: A Multidisciplinary Approach,
Robert Metzger, Elsevier Digital Press, 2004

The Worldview of the Literary Detective

One of the world's most popular forms of fiction is the detective novel, particularly the murder mystery. There are all kinds of detectives found in this genre. There are police professionals, private detectives, and avocational dabblers in detection. There are modern detectives and detectives from historical settings, all the way back to ancient Rome. Literary critics believe that the appeal of the murder mystery is the combination of the clear delineation of right and wrong and the intellectual challenge of identifying a villain.

What does all of this have to do with finding software defects? We can see an analogy between the murder mystery and the search for a software defect. The defect is considered a crime, and the programmer is the detective. A detective who wants to solve a crime needs answers to the following questions:

- Who did it?
- How did the culprit do it?
- When did the culprit do it?
- Why did the culprit do it?

We're interested in observing the thought processes and methodologies of these literary detectives. Most famous detective fiction is useless for the purpose of understanding ways to debug software. The authors may tell us about the quirky behavior of geniuses like Hercule Poirot or Nero Wolfe, but they never reveal how those detectives think through their cases.

To apply the wisdom of detective literature to debugging software, we must consider the analogy between software defects and crime. Our approach to motive is somewhat different, since we assume that all defects were caused accidentally. We are interested in an answer to the question, why did this problem happen? We treat the why question the way an accident investigator would, rather than the way a detective would. The detective seeks to assign guilt for a crime. The investigator seeks to find contributing causes, to prevent the accident from occurring again.

Detectives don't need to find a motive either, although juries are more easily convinced when a motive is given. If the prosecution presents an ironclad case that only one person had both the means and opportunity to commit a crime, a rational jury should convict the defendant. By analogy, if only one possible piece of code had both the means and opportunity to cause an observable failure, the programmer should focus on that code segment as the cause of the defect.

Sherlock Holmes

We begin our study of the way of the detective by investigating the career and philosophies of the greatest of all detectives in literature, Sherlock Holmes. We can summarize the points of Holmes's methodology that apply to software debugging under the following categories. In each case, at some point in one of his stories, Sherlock Holmes explains these methods. When we apply Holmes' principles to the process of debugging, we learn the following.

Use cross-disciplinary knowledge

When developing applications software, it's as important to understand the application as the software to be able to diagnose defects. Domain specific knowledge makes it possible to identify problems in logic that are plain to the user of the application.

Focus on facts

One of the fundamental tenets of modern science is that results must be reproducible to be accepted. Don't accept defect reports unless you're given sufficient data to reproduce the problem.

Pay attention to unusual details

To know what is unusual, you must know what is common. What should be common in an operational piece of software is correct behavior. Correct behavior is defined by an external reference.

Gather facts before hypothesizing

You shouldn't be experimenting unless you have a hypothesis you're trying to confirm. The best way to ensure that you have a hypothesis is to write it down.

State the facts to someone else

Describing a problem to someone else is one of the oldest ways programmers have used to uncover defects. Since the chief benefit of this method is to get the viewpoint of another person, the place to start is by giving your viewpoint.

Start by observing

Detecting and debugging differ in the extent to which we can create additional evidence. When collecting data, consider not only what to observe, but what point of view to observe from.

Don't guess

If you don't know what effect a change will have, you have no business making the change. A hypothesis includes a means of verifying the correctness of the proposition.

Eliminate impossible causes

Try to eliminate as many possible causes and culprits as you can with a single observation. Defects that are difficult to locate are often caused by a conjunction of several conditions, each of which isn't sufficient to cause the problem on its own.

Exclude alternative explanations

When you formulate a hypothesis, consider whether you already have evidence in hand that is qualifies the hypothesis. It takes less time to disqualify a hypothesis in this way than to perform yet another experiment.

Reason in both directions

Programmers almost always work on debugging inductively, rather than deductively. It is possible to reason from an observation to the causes for medium-sized systems, particularly if a single author created them.

Watch for red herrings

The English idiom “red herring” means a piece of information that distracts or deceives. When you come upon a fact that seems to bear upon your investigation, consider all of the interpretations of the fact.

Lord Peter Wimsey

Next, we consider the methods of the amateur sleuth Lord Peter Wimsey and how they can be applied to debugging software. We can summarize the points of Lord Peter's methodology that apply to software debugging under the following categories. In each case, at some point in one of his stories, Lord Peter Wimsey explains these methods. When we apply Lord Peter’s principles to the process of debugging, we learn the following.

Use alibis as clues

An alibi has three components: time, place, and corroboration. A program that has side effects has no alibi for the time a problem occurred or the place it occurred. The debugging equivalent of an alibi is the assertion that a given system component can't be the cause of a problem.

Exercise curiosity

Exercising curiosity is one way we leverage the labor of finding the cause of a defect. We use this curiosity when we investigate potential related problems, even before they're reported.

Reason based on facts

Facts aren't complete without qualification. Use the reporter's questions -- who, what, where, when, how often -- to qualify the facts of a defect report to make it useful.

Use the power of logic

If you believe that a phenomenon observable by the human senses is both true and false, you're going to have a hard time with the scientific method. All classical reasoning methods depend on this principle.

Enumerate possibilities

The trick to enumerating possibilities is knowing when you're done. You can enumerate possibilities by using the principle of noncontradiction and a binary tree.

Use a system for organizing facts

There are lots of good ways to organize a search for the explanation of a defect, but few programmers use any system. Some possible approaches include the black box, the scientific method, the reporter's method, and the tester's method.

Exercise caution when searching

In many disciplines we're told, don't make assumptions. Unfortunately, this is easier said than done. You can follow a method that minimizes the effects that assumptions have on your search.

Use gestalt understanding

If you're working with a large amount of data, display it visually so that your brain can recognize larger patterns you might not recognize from looking at numerical values. Color-coding items in a listing is an effective way to make your brain pick out patterns.

Show how something could be done

Use one of several diagram types to show how a defect could occur.

This book explores the methods for debugging organized to six intellectual disciplines. Each way as an analogy, a set of assumptions that forms a worldview, and a set of techniques associated with it. We follow a multidisciplinary approach, in which we seek the best methods for solving intellectual problems.