In Memoriam

William Robert Elmendorf
February 21, 1925 to August 31, 2006

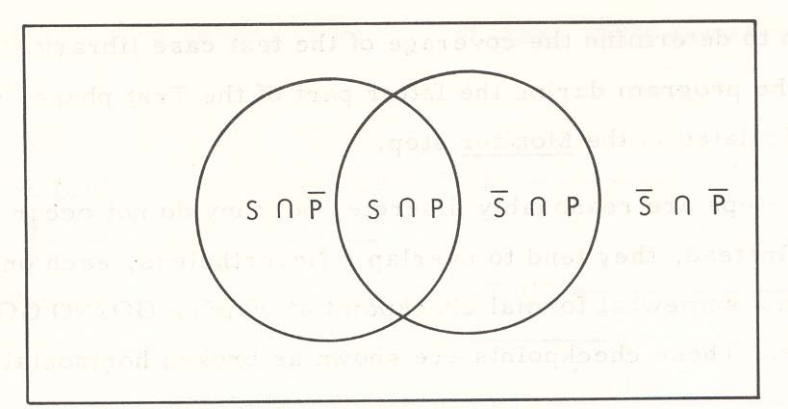
If you are a software tester there is someone you should know about – Bill Elmendorf. Bill is the father of disciplined, then rigorous, requirements based testing. He is the originator of such things as equivalence class testing and model based testing. I knew him for over thirty-five years so this is both a summary of his contributions to testing and a reflection on personal memories.

Bill received a Bachelors degree in Electrical Engineering from Cornell in 1949, worked for a few years, and then received a Masters in Accounting from the University of Colorado. His 1953 Masters thesis was entitled "Electronic Data Processing Machines in Accounting and Business". This was at a time when the head of IBM, Tom Watson, wrote that he only saw the need for a half dozen computers in the world. Bill joined IBM in 1953 at the Poughkeepsie, NY center, initially working on various hardware projects (by then Mr. Watson had re-evaluated his position). In 1963 Bill moved into software, first addressing programming languages and then the OS/360 architecture. In 1965, fortunately for the rest of us, he moved into software testing.

In June of 1967 Bill wrote one of the most important papers in the history of software testing – "Evaluation of the Functional Testing of Control Programs". He was working on how to test IBM's operating systems at the time. It is a thin document, only eight pages long, which initially saw only limited distribution. In it he describes the need for a disciplined approach to software functional testing and describes what that should be. [Note: A version of this paper was published in IEEE Transactions on Systems Science and Cybernetics Volume SSC-5, No. 4, October 1969, pp. 284-290.] Here are the key concepts from this paper:

1. "… a more scientific approach [to testing] is needed – one which overcomes the deficiencies of the 'testing-is-an-art' approach."

2. We need to test the software from a requirements perspective and from an internal perspective – i.e., you cannot test 100% of a system from the requirements given the nature of technology. His classic Venn diagram first appears in this paper.



Specification Testing Distinguished From Program Testing.

S stands for specified functions
P stands for programmed functions

Ideally the specified functions match the programmed functions. This is verified by specification and program testing. You look for specified functions not matched by programmed functions via specification based testing and programmed functions not matched by specified functions via program testing. You must also look for functions neither specified nor programmed but needed.

3. Test coverage should be weighted by such factors as "vulnerability to programming error, vulnerability to system failure, and market impact." – i.e., he is describing Risk Based Testing.

4. Functional variations should be defined by identifying the variables and the states to test for each variable. The states to test should include acceptable values, values at the extreme upper and lower acceptable values, and values beyond the extreme upper and lower acceptable values – i.e., equivalence class testing with boundary analysis.

5. Functional variations are then mapped into test cases resulting in a coverage matrix. This is used to track the test status.

6. Test Plans should be reviewed to ensure they are adequate – i.e., test the Test Plan.

7. Test progress should be tracked by the number of successful variations executed, not just the percentage of tests run successfully.

8. Regression testing must be blended with new function testing.

Today we take this all for granted. How else would you test? However, in 1967 this was a major advance on the state of the art.

By the time I joined IBM in 1969 all of this was the accepted practice in the labs. My first job there was as a tester in the Component Test group. Bill's office was down the hall from mine so naturally I wanted to be mentored by the master. He was always gracious and patient with my incessant questions.

In 1970 IBM formally recognized the major contributions Bill had made to testing by awarding him their Outstanding Contribution Award (OCA). The citation reads: "Disciplined Approach to Program Testing: Mr. Elmendorf contributed to the development process by structuring a disciplined approach to program testing, thus changing the control of functional testing from an art to a systematic process. Further evidence of this contribution is the general acceptance of this approach throughout IBM Programming Centers." Along with the OCA Bill received a bonus check for more than I was going to make that year. I remember sitting in the audience as he received the check (for an amount greater than my salary that year) thinking "testing is a very good profession to be in." I have been at it ever since.

Ironically, at the same time Bill was getting this award he had already determined that equivalence class testing was inadequate. All it did was reduce the total possible number of tests down to a manageable number of tests. It did not guarantee that all of the defects in the code would be detected. He noticed that hardware circuits had a very low residual defect rate for their functionality. He suggested that we should try to apply hardware logic testing techniques to testing software.

I was among a number of people asked to peer review the potential of this approach. I wrote a brilliant position paper "proving" that it would not work for software testing. I reasoned that hardware is all binary – ones and zeros / true and false. In software, on the other hand, if you add two thirty-two bit registers together you get 2^{64} combinations – QED it won't work. Bill totally ignored my paper and determined how to apply it. Not being a total idiot I managed to retrieve and destroy all copies of my "brilliant" paper. I also immediately began learning about this new approach.

In August of 1970 Bill documents this new technique in another seminal paper: "Automated Design of Program Test Libraries". The approach is based on the path sensitizing algorithms from hardware logic testing, specifically the D-algorithms. The major advance this offers is that any fault is propagated to a point where it becomes observable. Once again there are a number of fundamental breakthroughs to be found in this paper:

1. The first step in the process is to create a Cause-Effect Graph of the logic. This is the first application of Model Based Testing in software.

2. He recognizes that the act of graphing the specification actually is a test of the specification itself – i.e., we must test the specification, not just the code derived from the specification.

3. He describes how to create a set of tests that are necessary and sufficient to test the code from a black box perspective. The process eliminates redundancy in the test library while maximizing coverage.

4. He extends the concepts from the hardware version of the algorithms to cover software specific issues. The issue is constraints on the inputs – e.g., exclusive, one and only one. Today these are called preconditions and post conditions in defining use cases. The same concepts are also applied today in design by contract.

5. He introduces the concept of "Untestable" variations. These are variations which, due to the constraints and overall logic, cannot be sensitized to an observable point. The result is that you need to insert diagnostic probe points into the code in order to make sure that you got the right answer for the right reason. The process guides you to where you need to insert the diagnostics.

6. The test case design rules do not just address what input combinations to test. They also identify the expected results. This is the first test case design "Oracle".

7. He automates the test design process in a tool called TELDAP – TEst Library Design Automation Program. This is one of the first software test case design tools. It addresses the problem of ensuring that the set of tests are not only optimized but thoroughly documented.

When the process and the tool were applied to projects they resulted in a 40% to 50% reduction in the cost of the functional test effort while increasing coverage by 30% to 40%. This was as compared to equivalence class testing.

To underscore just how momentous Bill's breakthroughs are you need to understand the state of the art in testing at this time. In August of 1969 Bill produced yet another paper: "Program Testing – A Bibliography of Published Literature, 1962 – 1968". It is an exhaustive list of all testing papers published internally in IBM and externally from all the sources he could find including ACM, AFIPS, and IEEE. There are only 57 such papers identified. A number of them only address testing in a peripheral manner.

By 1974 I had left the IBM labs and gone back to the field. I began applying graphing to a wide variety projects as part of IBM's Contract Services Group (the predecessor to Global Services). In 1977 I formed my own consulting company with a primary focus on testing. Bill by this time had grown a bit frustrated with trying to get software people to be rigorous about testing and had gone back to the hardware side of IBM. We stayed in touch. I told him of project after project where we had successfully applied Cause-Effect Graphing in a wide variety of clients, even though we had no automated support.

In 1986 I convinced Bill to retire from IBM and join in partnership with me to create the next generation test design tool based on graphing. This would become SoftTest. We collaborated on the specifications. Bill wrote the code. I did the testing – of course using graphing. Now this was an interesting project for me – being the tester for the master of testing. Whenever I found a defect in his code the response was always the same. His reaction was totally positive. Any bug we found meant a user would never encounter it. When we shipped release one there were no defects reported by any customer.

In 1990 Bill decided to fully retire. After that I continually kept him up to date on new versions of the product, new clients using it, and the results from the projects. He got a great deal of pleasure and satisfaction in hearing about these efforts. Yet he never fully appreciated how significant his contributions to software testing were. In a resume he wrote for himself after retiring, covering a career of over forty years, he devotes only one line to testing: "Solely responsible for the development of a program which automatically defines functional variations and test libraries based on a formal description of a program's behavior". He had that rare blend of brilliance and humility.

Bill was my mentor, my partner, and my friend. My career has been built, to a great extent, on test approaches Bill pioneered. I owe him a huge debt of gratitude. If you are a software tester, so do you.

I want to thank Bill's wife Trudi, his son Doug, and his daughter Heidi for their gracious assistance in sharing a lot of material from Bill's personal papers.

Richard Bender
Bender RBT Inc.
rbender@BenderRBT.com
September 2006