Defect and Fault Seeding In Dependability Benchmarking

Barry Boehm, Daniel Port
University of Southern California
{boehm, dport}@sunset.usc.edu

Abstract

Defect and fault seeding is often considered for gathering empirical estimates within reliability models. Traditional defect seeding is fraught with difficult to resolve validity concerns when attempting to estimate true defect and fault populations. With dependability benchmarking we are less concerned about true defect and fault estimates, rather we wish to compare the relative effectiveness of dependability approaches. We propose that in this context the traditional concerns regarding defect and fault seeding techniques may not be as difficult to address and that potential new approaches may be useful as a means of benchmarking approaches to dependability.

1. Introduction

Within the context of traditional defect seeding suppose you use a dependability benchmarking capability to compare the performance of two tools on a system under test (SUT) – or its equivalent in code, design, or specification analysis – and both tools find the same 3 defects. This gives you a good comparative analysis of the tools, but leaves you wondering whether these three defects are 100% of the three remaining defects in the SUT, 10% of the 30 remaining defects, or something else.

Defect seeding approaches attempt to estimate the size of the defect population and the absolute effectiveness of defect detection techniques by deliberately introducing defects into a system. The general approach is:

- Insert N defects in the system under test (SUT).
- Run the tests, find M seeded defects, K unseeded defects.
- Estimate remaining defects as:

$$R= K *((N-M)/M), \text{ from } K/(K+R) = M/N.$$

Thus, if you seed the SUT with 10 defects and each tool also finds 6 of the 10 seeded defects, you can estimate that the 3 defects found by the tools are 60% of 5 previously-undetected defects in the SUT, and that two remain.

The defect seeding concept has been around since at least the early 1970's, but has fallen from practice because of difficulties in seeding defects in ways which satisfy the underlying assumptions of the estimation formula. These include:

1. The seeded defects are representative of existing defects. Seeding is mostly done by developers, whose blind spots miss many sources of defects.

2. The test profile is representative of the operational profile. Again, the developers' knowledge of actual usage patterns is generally highly imperfect.

3. The SUT is developed without knowledge of the seeding profile. If the seeded defects become well-known, there are risks of consciously or unconsciously tailoring the tool to look good on the seeded defect sample.

4. The source code available for defect seeding. As systems become increasingly COTS-based, this difficulty increases.

Individually and in concert, therefore, these assumptions are often invalid to some extent, leading to inaccurate estimates. However, there are not many strong alternative approaches available, and we feel it is worth exploring new approaches to defect seeding, which significantly strengthen the ability to satisfy these assumptions.

2. Potential New Approaches

We present several approaches that may help address some of the complications of the traditional approach to defect seeding (see [1] for details on several of these).

1. Change Histories. If the SUT is (say) version 3.4 of a given system, one can use fixes from earlier versions as sources of seeded defects. These have the representativeness advantage of having been real defects, but have the shortfall of having been the most detectable defects using current techniques. Also, the version changes may be complex combinations of defect fixes and

general upgrades, which makes preparing an appropriately-seeded SUT more difficult.

2. N-Version Programming. One can generate further representative defects by giving the specs to different programmers and generating a family of SUT versions. Comparative analysis of the defects found in the SUT versions can also generate estimates of the likely number of residual defects. Studies of N-version programming have shown that it is an imperfect source of independent implementations, and it can also be expensive, but it appears to be worth exploration. Program mutations are a similar source of defect-seeding alternatives.

3. Randomized Defect Seeding. One way to reduce the risks of gaming the seeding profile is to select random seeded defects from a large and/or parameterized sample. This also opens up the possibility of using multiple-run population estimators such as jackknife and bootstrap methods.

4. Use of Defect Distribution Statistics. One can also combine randomized defect seeding with defect distribution statistics to address the defect representativeness issue. Orthogonal Defect Classification statistics are a good example.

5. Connectors and Wrappers. One can deal with COTS defect and fault seeding to some extent by using connectors or wrappers to simulate potential real faults. Examples are data corruption faults via wrapper modifications of the output stream, or uses of connectors to generate communication failures (timing, handshaking, noise, etc.)

3. Issues for Discussion

Some issues worth exploring at the Workshop include:

- Mapping of preferred defect seeding approaches to dependability attributes. These include: Robustness (reliability, availability, survivability), Protection (security, safety), Quality of Service (accuracy, fidelity, performance assurance), and Integrity (correctness, verifiability).

- Alternative concepts and approaches. These could include mutation testing as defect seeding; model-driven approaches; information theoretic approaches; or game theoretic approaches.

- Special application challenges, such as scalability, test oracles (e.g., for agent-based systems of systems), and Heisenbug effects (additional defects induced by seeded defects, such as timing and synchronization defects).

4. References

[1] Voas, J., McGraw, G., *Software Fault Injection*, Wiley, 1998.