IV & V is not V & V so how does that change things? *

Tim Menzies*, Robert (Mike) Chapman*

* Computer Science, Portland State University, USA tim@barmag.net URL: http://barmag.net
* Galaxy Global Corporation Robert.M.Chapman@ivv.nasa.gov

Abstract: IV & V is not V & V. IV & V activities are opportunistic stochastic samples of project artifacts, some of which may be incomplete and imprecisely described. The mathematics of systems shows that such sampling can be surprisingly effective. The best tools for IV & V understands how to perform cost-effective sampling, and can alert us when that sampling process is incomplete and needs to be repeated.

Contents

1 Introduction ........................................... 1
2 The Data Drought ...................................... 2
3 A Model of IV & V ..................................... 2
4 Mathematics of Probing ............................... 2
5 So, is IV & V Possible? .............................. 2
6 IV & V Tools ......................................... 3
   6.1 About ONION .................................. 4

1 Introduction

In V & V, developers assess their own code. In IV & V, the developer’s code is assessed by someone else.

We need to better understand what is unique about IV & V. As more and more software development goes off-shore, IV & V will become more frequent as (e.g.) American-based companies strive to oversee software developed at other site.

NASA in particular needs to understand IV & V better. A large network of contractors build NASA systems yet if any of those systems fail, the general public blames “NASA” and not “contractor XYZ”. Such failures severely damage the prestige of the agency and its ability to collect funding for future endeavors. Hence, it is vital the agency as a whole institutes a sampling method for all those contractor systems. This sampling method can only be heuristic since a complete sampling policy would be prohibitively expensive. For software, the current sampling policy is managed by the Fairmont software IV & V team.

Despite this growing dependency on IV & V, we don’t see in the literature a recognition of the essential differences between V & V and IV & V. For example, while the formal methods community improve their ability to automatically find how systems can violate global temporal constraints, we would classify those techniques as V & V not IV & V since they require detailed knowledge of the system. As we shall see, my characterization of IV & V is software assessment despite a poverty of data about this system.

Much of work is empirical and empirical research needs data. The data used for the case study at the end of this paper is all public domain and is freely available at the NASA Metrics Data Program web site (see Figure 1).

The rest of this paper is structured as follows. After some notes on the data drought, we propose a novel model of IV & V. This is followed by some notes on the mathematics of prob-

Fig. 1 The MDP data repository http://mdp.ivv.nasa.gov.

* This research was conducted at Portland State University and the NASA Software IV & V Facility, Fairmont, West Virginia, under various NASA contracts including the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV & V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.
ing a system and some empirical and theoretical results suggesting that the internals of a program may be surprisingly simple. That math, and those results, has inspired a suite of new IV&V tools. V&V teams need in-depth tools that seek (say) six sigma certification of a system. Our new IV&V tools are different. They are not in-depth tools; rather, they are in-breadth tools that try to survey a wide-sample of behaviors in a system in a cost-effect manner.

2 The Data Drought

We base this paper on our time at NASA's IV&V group. Since last century, we have worked or with NASA's IV&V team in Fairmont West Virginia. In that role, we watched numerous attempts to build rigorous work-breakdown structures (WBSs) of IV&V. We came to believe that traditional prescriptive structures (e.g. IEEE8012) were inappropriate to describe IV&V. For one thing, we could not even assess the value of those WBS, for reasons that are insightful to the process of IV&V. IEEE8012 lists dozens of activities and we could only ever find data on a few of them.

This data drought was a persistent problem. V&V teams interact with local developers while they are actively working on some sub-system. IV&V teams, on the other hand, interact with remote developers who may have moved on to other tasks. Hence, by definition, V&V can access more information more quickly than an IV&V team. An accurate model of IV&V must therefore work despite not having full access to all details of a project.

3 A Model of IV&V

Since we couldn't use some IEEE8012-variant to describe IV&V, we turned to a more opportunistic model. My preferred model is a stochastic and opportunistic manner. In this model, an IV&V agent gets sent randomly selected project artifacts. The agent is required to perform some value-added analysis on the artifact.

Our IV&V agent faces three challenges. Firstly, the skills and experience of an IV&V agent may vary wildly. For example, rarely does a IV&V agent have detailed and expert knowledge about the domain of the system.

Secondly, IV&V is usual a resource-bound activity (e.g. 5% if the total system cost). Hence, our IV&V must works under strict budgetary constraints.

Thirdly, the artifacts are documented to a varying standard and so, in the usual case, they are not unambiguous; not complete; not consistent; and, not up-to-date with latest changes to a project. Our agent therefore has to work in an opportunistic manner:

- Applies whatever tools\(^1\) they have experience; . . .
- . . . to whatever artifacts they have from the projects; . . .
- . . . to generate a list of anomalies that requires further exploration.

The nature of those anomalies and feedback from the project determines what happens next. For example, the project could offer new information that demonstrates that the anomalies are not an important issue. Alternatively, the project may reject the anomalies as unimportant in which case the IV&V agent has to apply other methods to confirm or deny that the anomalies indicate that some rework is required.

Hence IV&V can't be prescribed in some one-size-fits-all process document (e.g. some IEEE8012-variant). IV&V should never be routine. Our IV&V agents should never congratulate themselves for completing a checklist. Rather, IV&V should be all about the surprise, the exceptional, and the hunt for real errors within the space of detected anomalies.

Can we trust such a surprise-driven opportunistic and stochastic process to assure that the system is being developed properly, and that the right system is being developed or acquired? The answer is "No" and anyone who says otherwise does not understand the mathematics of probing a system.

4 Mathematics of Probing

Consider a system containing \(V\) variables with \(S\) assignments may require one test for each combination of assignments; i.e. \(#\) tests \(= N = S^V\). For example, consider one sample of fielded expert systems in which knowledge bases were found to contain between 55 and 510 "literals" [13]. Literals offer two assignments for each proposition: true or false; i.e. \(S = 2\) and \(V\) is half the number of literals. Assuming:

- It takes one minute to consider each test result (which is a gross underestimate), and
- The effective working year is 225 six hour days,

then a test of those sampled systems would take between 29 years and \(10^{70}\) years (a time longer than the age of this universe).

For another example consider how a linear increase in the confidence \(C\) that we have found all defects can take exponentially more effort. For one-in-a-thousand detecting, moving \(C\) from 90% to 98% takes 2301 to 3910 black box probes (respectively)\(^2\). That is, increasing \(C\) by just under 10% requires nearly double the tests.

5 So, is IV&V Possible?

Given such a large space of possibilities, how can we assure that any system behaves as expected over the space of

---

\(^1\) Here "tool" means either some automatic process or some manual technique such as manual inspection.

\(^2\) To see that, recall that a randomly selected input to a program will find a fault with probability \(x\). After \(N\) random black-box tests, the chances of the inputs not revealing any fault is \((1 - x)^N\). Hence, the chances \(C\) of seeing the fault is \(1 - (1 - x)^N\) which can be re-arranged to \(N(C, x) = \frac{\log(1 - C)}{\log(1 - x)}\). For example, \(N(0.90, 10^{-3}) = 2301\) [14].
its possible states? The answer is that most systems contain far fewer than \(S^{X+Y}\) reachable states.

A lower bound on the number of internal states of a system is the number of states of the input variables. This lower bound may be very small indeed. For example, Avritzer et al. [1] studied the 857 different inputs seen in 355 days operation of an expert system. Massive overlap existed between these input sets. On average, the overlap between two randomly selected inputs was 52.9%. Further, a simple algorithm found that 26 carefully selected inputs covered 99% of the other inputs while 53 carefully selected inputs covered 99.9% of the other inputs.

More generally, studies on the operational profile of a program's input parameters suggest that the input space of a program may be be particularly complex or large. Figure 2 shows a study where the original operational profile \(OPo\) was compared to three profiles containing an increasing number of errors. The mutants were called (in order of increasing errors) \(OP1\), \(OP2\), \(OP3\). The inaccuracies in the operational profiles were very apparent after a small number of tests. However, above 100 tests randomly selected from each profile, the errors of the different profiles converged.

Druzdell offers a simple model that explains why there are so few internal reachable states [3]. If software has \(n\) variables, each with its own assignment probability distribution of \(p_i\), then the probability that a system will fall into a particular state is \(p = p_1p_2p_3\ldots p_n = \prod_{i=1}^{n} p_i\). By taking logs of both sides, this equation becomes

\[
\ln p = \ln \prod_{i=1}^{n} p_i = \sum_{i=1}^{n} \ln p_i
\]

The asymptotic behavior of such a sum of random variables is addressed by the central limit theorem. In the case where we know very little about software, \(p_i\) is uniform and many states are possible. However, the more we know about a system the more varied are individual distributions. Given enough variance in the individual priors and conditional probabilities or \(p_i\), then the expected case is that frequency with which we reach states will exhibit a log-normal distribution; i.e. a small fraction of states can be expected to cover a large portion of the total probability space; and the remaining states have practically negligible probability.

Druzdell found a log-normal distribution in the frequency of reached states within one system he instrumented: see Druzdell

6 IV & V Tools

The above line of reasoning has motivated the development of a new suite of IV & V tools: LURCH, TAR3 and ONION. These are IV & V tools; i.e. they search for the features that the V & V team might of missed. Since they must work in data-starved domains, we assess these tools on traditional means as well as how well they perform on minimal data. For example, LURCH is a rapid Monte Carlo method for formal models where some of the details of the models are unknown [10]. LURCH treats such models as a random distribution of the space of many models. This space is rapidly sampled and consistent subsets of each one is generated.
Fig. 4 A ROC sheet assessing the detector $v(g) \geq 10$. Each cell \{A,B,C,D\} shows the number of modules, and the lines of code associated with those modules, that fall into each cell of this ROC sheet.

One advantage of LURCH's search is that its search is stochastic. Hence, LURCH can ignore any incorrect preconceptions of system behavior from the development team, we often used a stochastic search. Further, given the state space clustering results shown above, such a stochastic search can be remarkably effective at exploring the reachable parts of a system. In an observation consistent with Druzdal's results, LURCH can often find as many violations to system temporal properties than more complete searches, and do so in a fraction of the memory [11].

LURCH generates a lot of Monte Carlo data and TAR3 summarizes that data using treatment learning. TAR3 is a minimal contrast set learner that returns the difference between Monte Carlo outputs with different scores [7]. Often, TAR3 often finds minimal contrast sets i.e. a small number of outputs are adequate for separating preferred and undesirable outcomes. TAR3 extends the Druzdal results as follows. Druzdal argued that, in the expected case, there are a surprisingly small number of reachable states. The presence of minimal contrast sets that selects for different states suggests that there may be a small number of control variables that select for different states.

ONION has not been reported previously and is being tested for software defect detection.

6.1 About ONION

Standard practice is to apply the best available assessment methods on the sections of the program that the best available domain knowledge declares is most critical. We endorse this approach. Clearly, the most critical sections require the best known assessment methods. However, this focus on certain sections can blind us to defects in other areas. If most of the assessment effort explores project artifacts A,B,C,D, then that leaves a blind spot in E,F,G,H,I,.... Therefore, standard practice should be augmented with a lightweight sampling policy to explore the rest of the system.

One common lightweight sample policy is to use defect detectors learned from static code measures. Data miners input logs of such measures, plus the logs of known past defects for those modules, to generate predictors for future defects. Such detectors can be assessed via their accuracy; i.e.

\[
\text{band } B_{v_{c}} \quad \text{range}
\]

<table>
<thead>
<tr>
<th>band $B_{v_{c}}$</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{1}$</td>
<td>0.84 &lt; $T &lt; 1.73$</td>
</tr>
<tr>
<td>$B_{2}$</td>
<td>1.00 &lt; $T &lt; 5.89$</td>
</tr>
<tr>
<td>$B_{3}$</td>
<td>5.67 &lt; locCodeAndComment &lt; 19.19</td>
</tr>
<tr>
<td>$B_{4}$</td>
<td>locCodeAndComment &gt; 19.19</td>
</tr>
<tr>
<td>$B_{5}$</td>
<td>5.15 &lt; branchCount &lt; 8.90</td>
</tr>
<tr>
<td>$B_{6}$</td>
<td>locComment &gt;= 56.38</td>
</tr>
<tr>
<td>$B_{7}$</td>
<td>locComment &lt; 7.67</td>
</tr>
</tbody>
</table>

Fig. 5 Extracted bands

the number of true negatives and true positives seen over all events. In terms of the cells \{A,B,C,D\} shown in Figure 4, accuracy is $\text{Acc} = \frac{A+B}{A+B+C+D}$.

Apart from accuracy, several other measures are of interest. The probability of detection, or "PD", is the ratio of detected signals, true positives, to all signals; i.e. $PD = \frac{D}{B+D}$ (PD is also called the recall of a detector). Also, the probability of a false alarm, or "PF", is the ratio of detections when no signal was present to all non-signals; i.e. $PF = \frac{C}{A+C}$.

Another statistic of interest is the effort associated with a detector. If the detector is triggered, then some further assessment procedure must be called. For the particular static code defect detectors discussed in this paper, we will assume that this effort is proportional to the lines of code in the modules. Under that assumption, the effort for a detector is what percentage of the lines of code in a system is selected by a detector; i.e. $effort = E = \frac{LOC_{c} + LOC_{a} + LOC_{r} + LOC_{n}}{LOC_{c} + LOC_{a} + LOC_{r} + LOC_{n}}$.

For years, with DiSefano and Chapman and Orosco [6,8, 9], we have strived to generate detectors have high POS, low PFs, and low effort. This ideal state rarely happens. The only way to make no mistakes (and achieve a PF of zero) is to do nothing which in turn, means that nothing will be detected (i.e. $PD=0$). Also, the only way to catch more defects is to make more mistakes (i.e. increasing PD means increasing PF). Further, PD and effort are linked: the more modules that trigger the detector, the higher the PD. However, effort also gets increases.

Standard data miners, built to optimize accuracy, generally generate detectors with effort higher than PD. Recent results with a novel rule-covering algorithm called ONION have been more promising. These results are preliminary but encouraging. ONION divides all numeric attributes into, say, ten bands. The band with the highest ratio of $pd eff_{total}$ is then printed to the screen. All the modules with that band are then removed from the defect log and the algorithm repeats on the remaining data.

The results are the two reports in Figure 5 and Figure 6. Figure 5 shows the order of the nine bands $B_{1} \ldots B_{9}$ found by ONION in a particular NASA defect log (1, T, B, N, etc) are Halstead measures). Figure 6 shows a set of "what-if" queries where a composite defect detector was built from a disjunction of the first $N = 1, 2, \ldots, 9$ bands.

From Figure 6, an IV&V agent could select a defect detector that matches the available effort and meets the desired effort and $pf$. Note that a disjunction of the first three bands
<table>
<thead>
<tr>
<th>&quot;OR&quot; the first ( N ) bands</th>
<th>pf</th>
<th>acc</th>
<th>pd</th>
<th>effort</th>
<th>( \frac{pd}{effort} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>18</td>
<td>80</td>
<td>55</td>
<td>33</td>
<td>1.65</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>83</td>
<td>49</td>
<td>27</td>
<td>1.80</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>83</td>
<td>48</td>
<td>26</td>
<td>1.85</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>83</td>
<td>44</td>
<td>22</td>
<td>2.02</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>83</td>
<td>40</td>
<td>18</td>
<td>2.29</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>89</td>
<td>30</td>
<td>12</td>
<td>2.40</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>89</td>
<td>23</td>
<td>7</td>
<td>3.45</td>
</tr>
</tbody>
</table>

Fig. 6 Effects of "OR"-ing together the first \( N \) bands (shown on the last line of Figure 6) has a \( PD \) over three times higher than \( effort \). To the best of our knowledge, no other defect detection method has generated detectors with \( pd \gg effort \).

References