Innovations Syst Softw Eng
DOI 10.1007/s11334-008-0046-3

ORIGINAL PAPER

Learning better IV&V practices

Tim Menzies · Markland Benson · Ken Costello · Christina Moats · Melissa Northey · Julian Richardson

Received: 3 November 2007 / Accepted: 10 January 2008

Abstract After data mining National Aeronautics and Space Administration (NASA) independent verification and validation (IV&V) data, we offer (a) an early life cycle predictor for project issue frequency and severity; (b) an IV&V task selector (that used the predictor to find the appropriate IV&V tasks); and (c) pruning heuristics describing what tasks to ignore, if the budget cannot accommodate all selected tasks. In ten-way cross-validation experiments, the predictor performs very well indeed: the average f-measure for predicting four classes of issue severity was over 0.9. This predictor is built using public-domain data and software. To the best of our knowledge, this is the first reproducible report of a predictor for issue frequency and severity that can be applied early in the life cycle.

Keywords IV&V · Data mining · Early life cycle defect prediction · NASA

1 Introduction

All software has defect issues, some of which are seen by developers. We seek a predictor for the *number* and *severity* of issues that a project will produce. If applied *early* in the life cycle then such a predictor could check if extra funding is required for project verification and validation (V&V).

Wallace and Fujii [1] distinguish:

- *Embedded V&V*, performed in-house by developers;
- *Independent V&V* (IV&V), performed by an external team that, potentially, explores very different issues to the embedded V&V team.[1]

Previously, we have offered guidance to IV&V teams via issue predictors learned from static code measures [3]. Such predictors can be used only *after* the code has been written. Since it is much cheaper to fix errors *before* code generation [1,4–7], this article seeks early life cycle issue predictors.

We show here that a predictor for issue frequency and severity can be built in a *reproducible* way from public-domain data[2] and software.[3] The predictor is *operational*,

This research was conducted at West Virginia University and the NASA IV&V Facility under NASA subcontract project 100005549, task 5e, award 1002193r. 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.
See http://menzies.us/pdf/07ivv.pdf for an earlier draft.

T. Menzies (✉)
Lane Department of Computer Science and Electrical Engineering,
West Virginia University, Morgantown, WV, USA
e-mail: tim@menzies.us

M. Benson · K. Costello · C. Moats · M. Northey · J. Richardson
NASA's IV&V Facility, Fairmont, WV, USA
e-mail: Markland.J.Benson@nasa.gov

K. Costello
e-mail: Kenneth.A.Costello@nasa.gov

C. Moats
e-mail: Christina.D.Moats@nasa.gov

M. Northey
e-mail: Melissa.S.Northey@nasa.gov

J. Richardson
Powerset Inc., San Francisco, USA
e-mail: julianrichardson@acm.org

[1] Annex C of the IEEE 1012-2004 [2] standard offers a more detailed definition of IV&V, which retains Wallace & Fujii's essential distinction between internal (embedded) and external (independent) V&V.

[2] http://promisedata.org/repository/data/mb2.

[3] The Waikato environment for knowledge analysis (WEKA) data mining workbench [8].

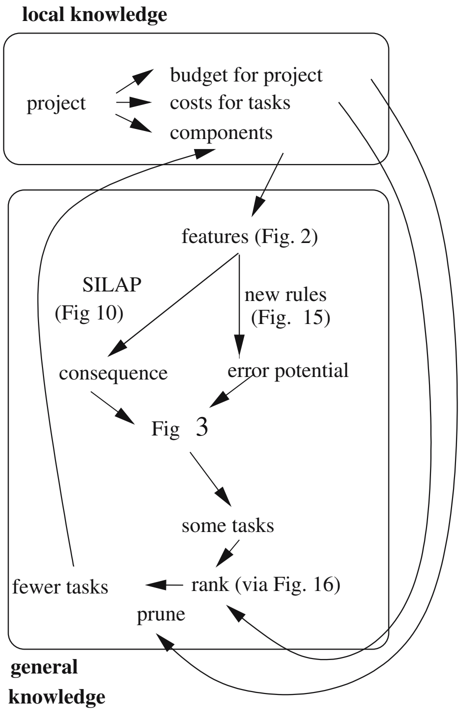

Fig. 1 SILAP2

i.e., it can be executed to generate precise statistics on its performance. For example, we show below that for National Aeronautics and Space Administration (NASA) independent verification and validation (IV&V) data, the model is remarkably effective at distinguishing between projects that will or will not produce numerous high-severity risks. Since the features used in our predictor can be collected early in a project's life cycle, this method can be quickly applied to new projects. To the best of our knowledge, this is the first report of an issue frequency and severity predictor that:

- is based on public domain data
- has a precise operational implementation
- has a demonstrably useful performance
- can be tuned to local data
- most importantly, can be applied early in the life cycle

Our proposed method extends the *software integrity level assessment process* (SILAP), an existing NASA IV&V task selection method [9,10]. As shown in Fig. 1, SILAP2 requires some *local knowledge* of a project: specifically, (a) the total *budget*; (b) the *costs* associated with applying various IV&V *tasks* which depends on domain details such as the skills and fees of subcontractor, etc.; and (c) the project *components*, i.e., large sections of the project that are often treated separately to the whole. This local knowledge is then mapped into the *general knowledge* described in this paper. Each project *component* is described using the *features* described in Fig. 2.

Derived	Raw features
co = Consequence	am =Artifact Maturity
dv = Development	as =Asset Safety
ep = Error Potential	cl =CMM Level
pr = Process	cx =Complexity
sc = Software Characteristic	di =Degree of Innovation
	do =Development Organization
	dt =Use of Defect Tracking System
	ex =Experience
	fr =Use of Formal Reviews
	hs =Human Safety
	pf =Performance
	ra =Re-use Approach
	rm =Use of Risk Management System
	ss =Size of System
	uc =Use of Configuration Management
	us =Use of Standards

Fig. 2 SILAP turns *raw features* into *derived features*

Each feature is scored using the scales described in the appendix. The features are then converted to *error potential* and *consequence* scores using SILAP and the rules found by this paper. These scores are then passed to the task selection table of Fig. 3, which proposes a set of *tasks* for each *component*. Each proposed task can be *cost*ed (via local knowledge) and *rank*ed (via historical records of task cost effectiveness). If the available *budget* is insufficient to complete all the *tasks*, then some can be *pruned* away.

The rest of this paper is structured as follows. Before presenting methods for tuning IV&V to particular projects, we make two digressions. First, using a wide range of formal and informal evidence, we make the case that IV&V is worthy of study. Second, a literature review will explore what little is known of current IV&V tasks. This will be followed by a general discussion of data mining methods for software engineering (SE) data and a specific discussion of what was learned by those data miners from NASA IV&V data. The paper ends with some comments on external validity.

2 Benefits of IV&V

There is much formal and informal evidence for the value of IV&V (by informal evidence, we mean that evidence that is *not* published in sources that other researchers can verify). Interviews with NASA IV&V personnel has yielded the following informal evidence for the benefits of IV&V:

- "Even when developers assess their own products, they focus on a very small part of a much larger system. IV&V workers, on the other hand, can explore a larger area."
- "IV&V has become NASA's corporate memory for software-related issues. Development teams can change personnel faster than the IV&V team so IV&V can come to understand the software as well or better than the development team. For example, in several NASA multi-year projects, most of the issues were found by the IV&V team, and not the developers. Also, in numerous cases,

			consequence					error potential				
phase	wbs	factor	1	2	3	4	5	1	2	3	4	5
concept	2.1	Reuse Analysis⋆			X	X	X					
	2.2	Software Architecture Assessment			X	X	X					
	2.3	System Requirements Review			X	X	X				X	X
	2.4	Concept Document Evaluation				Z	Z					Z
	2.5	SW/User Requirements Allocation Analysis				Z	Z					Z
	2.6	Traceability Analysis				Z	Z					Z
requirements	3.1	Traceability Analysis - Requirements		X	X	X	X				X	X
	3.2	Software Requirements Evaluation			X	X	X				X	X
	3.3	Interface Analysis - Requirements				X	X			X	X	X
	3.4	System Test Plan Analysis			X	X	X					
	3.5	Acceptance Test Plan Analysis					X					
	3.6	Timing and Sizing Analysis									Z	Z
design	4.1	Traceability Analysis - Design							X	X	X	X
	4.2	Software Design Evaluation				X	X				X	X
	4.3	Interface Analysis - Design					X				X	X
	4.4	Software FQT Plan Analysis		X	X	X	X					
	4.5	Software Integration Test Plan Analysis									X	X
	4.6	Database Analysis								X	X	X
	4.7	Component Test Plan Analysis										X
implementation	5.1	Traceability Analysis - Code					X		X	X	X	X
	5.2	Source Code and Documentation Evaluation				X	X			X	X	X
	5.3	Interface Analysis - Code				X	X			X	X	X
	5.4	System Test Case Analysis				X	X					
	5.5	Software FQT Case Analysis				X	X					
	5.6	SW Integration Test Case Analysis										X
	5.7	Acceptance Test Case Analysis					X					
	5.8	SW Integration Test Procedure Analysis										X
	5.9	SW Integration Test Results Analysis									X	X
	5.10	Component Test Case Analysis										X
	5.11	System Test Procedure Analysis					Z					
	5.12	Software FQT Procedure Analysis					Z					
test	6.1	Traceability Analysis - Test		X	X	X	X					X
	6.2	Regression Test Analysis									Z	Z
	6.3	Simulation Analysis					Z					
	6.4	System Test Results Analysis				X	X					
	6.5	Software FQT Results Analysis				X	X					
other	7.1	Operating Procedure Evaluation					Z					
	7.2	Anomaly Evaluation					Z					
	7.3	Migration Assessment					Z					
	7.4	Retirement Assessment					Z					

Fig. 3 SILAP's IV&V tasks. For definitions of each task, see [11]. For example, task 2.3 (System Requirements Review) is selected if consequence is greater than 2 *or* the error potential is greater than 3. Tasks marked with "Z" are only for human-rated flights. The *asterisk* on task 2.1 denotes a task that it selected when RA (reuse approach) is greater than 1. Software phases are numbered according to a NASA convention, according to which phase 1 is reserved for management tasks conducted across the life cycle, whereas phases 2, 3, 4, 5, and 6 represent concept, requirements, design, implementation, and test, respectively. *FQT* formal qualification tests

the IV&V team can recall why (e.g.) 3 years ago, some crucial decision was made."

- "The technical skill of NASA's IV&V's teams are acknowledged by (some) of NASA's contractors. In one case, the prime contractor voluntarily gave up funding within its own budget so that the NASA project office could pay for more IV&V, over and above the amount originally planned for IV&V."

More formal evidence for the benefit of IV&V comes from the literature and NASA technical reports. Zelkowitz and Rus discuss the Space Shuttle IV&V program in the 1990s [12]. They report a large decrease in major flight defects once IV&V was introduced, and that IV&V uncovered and resolved certain shortcomings in the shuttle's testing program. Almost half the issues found by IV&V were found very early in the life cycle of shuttle software upgrades.

Figure 4 shows the issues found in recent NASA missions by the IV&V team. Note that (a) most of the issues are found early in the life cycle (in the concept and requirements phase) and (b) many of the issues were of high criticality (severity 1 and 2), especially for the human-rated missions.

Economically, it is useful to find errors early. Boehm and Papaccio advise that reworking software is far cheaper earlier in the life cycle than later "by factors of 50–200" [4].

This effect has been widely documented in other research. A panel at IEEE Metrics 2002 concluded that finding and fixing severe software problems after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase. [6]. Also, Arthur et al. [7] conducted a small controlled experiment where a dozen engineers at NASA's Langley Research Center were split into development and IV&V teams. The same application was written with and without IV&V. Figure 5 shows the results:

Fig. 4 A sample of issues found by NASA's IV&V team

phase	S = severity	F_S = number of issues found of severity S: 9 robotic missions	7 human-rated missions
2. concept	1	0	59
	2	7	81
	3	54	168
	4	11	0
	5	2	0
	$\beta_2 = \sum_{S=1}^{5} F_S \cdot 10^{5-S}$	12,521	687,800
3. requirements	1	1	291
	2	70	313
	3	874	430
	4	435	0
	5	133	0
	$\beta_3 = \sum_{S=1}^{5} F_S \cdot 10^{5-S}$	171,883	3,266,000
4. design	1	0	11
	2	18	16
	3	275	96
	4	129	0
	5	25	0
	$\beta_4 = \sum_{S=1}^{5} F_S \cdot 10^{5-S}$	46,815	135,600
5. implement	1	1	51
	2	59	78
	3	364	333
	4	282	0
	5	76	0
	$\beta_5 = \sum_{S=1}^{5} F_S \cdot 10^{5-S}$	108,296	621,300
6. test	1	1	15
	2	10	27
	3	361	77
	4	271	0
	5	41	0
	$\beta_6 = \sum_{S=1}^{5} F_S \cdot 10^{5-S}$	58,851	184,700

phase	IV&V	no IV&V
requirements	16	0
high-level design	20	2
low-level design	31	8
coding & user testing	24	34
integration & testing	6	14
totals	97	58

Fig. 5 Defects found with and without IV&V, from [7]

		Phase issue found					
i	Phase issue introduced	f=1 Requirements	f=2 Design	f=3 Code	f=4 Test	f=5 Int	f=6 Operations
1	Requirements	1	5	10	50	130	368
2	Design		1	2	10	26	74
3	Code			1	5	13	37
4	Test				1	3	7
5	Integration					1	3
Δ = mean	$\frac{C[f,i]}{C[f,i-1]}$		5	2	5	2.7	2.8

Fig. 6 $C[f, i]$: the cost-to-fix escalation factors, relative to fixing an issue in the phase where it was found (f) versus in the phase where it was introduced (i). *Last row* shows the cost-to-fix delta if the issue introduced in phase i is fixed immediately afterwards in phase $f = i+1$, from [5]

(a) more issues were found using IV&V than without; (b) the issues were found much earlier. This is a compelling argument for IV&V. Even if the IV&V team found the *same* bugs as the development team, but found them *earlier*, the cost-to-fix would be reduced by a significant factor.

Similarly, Wallace and Fujii [1] discuss studies on IV&V started at different phases. Starting at the coding phase saves 20% of the V&V costs. However, starting earlier at the requirements phase saves 92–180% of the V&V costs.

Finally, Fig. 6 shows the cost of quickly fixing an issue relative to leaving it for a later phase (data from four NASA projects [5]). The last line of that table shows that delaying issue resolution even by one phase increases the cost-to-fix of $\Delta = 2, \ldots, 5$. Using this data, Dabney et al. [5] calculates a dollar spent on IV&V returns to NASA $1.21, $1.59, $5.53, and $10.10 (on four NASA projects).

3 Related work

Having documented the benefits of IV&V, the next question a manager might ask is how to do it? This is a difficult question to answer. The literature describes nearly a 100 V&V and IV&V tasks:

- Wallace and Fujii offer 20 essential and 36 optional tasks for software V&V [1].
- Table 2 of the IEEE-1012 standard for software V&V offers a list of 52 minimal tasks for IV&V. Each task is selected by a combination of development phase and estimated severity in project errors [2].
- Figure 3 offers 40 predeployment IV&V tasks.

In the literature, there are very few attempts to assess the relative cost-effectiveness of this broad range of IV&V techniques empirically. Most of the reports take the form of (e.g.) Easterbrook et al. [13] who discuss the merits of using a single lightweight formal method for IV&V requirements models on one NASA project. Another study worthy of mention is that of Hayes et al., who explore the relative merits of different tools for finding links between requirements documents [14]. That analysis includes a comparison of human expert and state-of-the-art commercial tools and a new research prototype. For more examples of singleton evaluations of one method on one project, see the proceedings of the annual symposium program[4] and the repository of NASA's software assurance research.[5]

Several researchers offer detailed taxonomies of standard NASA faults [15,16], but do not address what tasks are most cost-effective at finding the faults. For example, Lutz reports studies on the most frequent kind of faults seen in deep-space NASA missions but does not fully address the issue of what V&V and IV&V tasks are best at removing faults [17].

Leveson talks in general terms about how to increase the safety of software [18]. Her preferred methods would significantly change current IV&V practice [19] so it is difficult to map that advice into selecting the IV&V tasks of Fig. 3.

In a limited number of publications, some attempt is made to generalize over multiple projects, multiple methods, or projects extending over many years. Those publications include:

- Madachy's heuristic software risk model that alerts when certain observations are seen in a project [20];
- Boehm and Basili's top-10 defect reduction list [6,21];
- The Dabney study, based on four NASA projects [5];
- Menzies et al.'s discussion of learning predictors for software defects using data from five NASA projects [22,23]; or methods for generating cost models from 25 software projects, including 11 NASA developments [24];
- Glass's list of six factors seen in 16 runaway software projects; i.e., projects where "schedule, cost, or functionality that was twice as bad as the original estimates" [25];
- Raffo et al.'s software process of a standard waterfall model, modified to handle IV&V tasks [26].
- Jiang et al.'s table of the connection of 40 project features to the software risks seen in 83 projects [27].
- Ropponen and Lyytinen analysis of questionnaires from 83 project managers and 1,110 projects that identifies 26 software risk components [28, p110];
- Takagi and Abe et al.'s study of 31 projects that identifies features that predict for software runaways [29,30].

For our purposes, these studies are incomplete. Some of this research bases its conclusions on just a few projects [5,22,23, 25]; Others use a Delphi-style analysis were the authority of the conclusions rests on the authority of the author [6,20,21]. Alternatively, they may base their conclusions on extensive data collection but, for confidentiality reasons, cannot make that data available for public scrutiny [26–28]. It is hard to make critical audit conclusions based on a Delphi-style analysis or inaccessible data. In general, only a minority of SE researchers can publish the data used to make their conclusions (for example, [22–24,29,30]).

Also, several of the these papers [27,28] only offer general advice about how to avoid problems in software development. The subjective nature of this advice makes it difficult to consistently deploy them over a national software development program. Only a few [23,24,29,30] offer public-domain versions of their data and models.

Note that, even in combination, the above publications cover a small fraction of the (I)V&V tasks listed in the literature.

4 Data mining

Since existing sources were not informative, this study applied data mining to 40 IV&V tasks used in 211 NASA components from six projects.

4.1 Issues with data collection

It may surprise the reader that after 19 years, this study could only access 211 components. After all, as shown in Fig. 7, NASA has a 19-year history of IV&V.

What must be understood is that NASA is similar to many other large organizations that constantly adapt their practices and their databases as their business changes. Various factors have resulted in a steady pace of change at NASA:

- In the 1990s, NASA moved to a "faster, better, cheaper" paradigm that led to massive changes in NASA's development practices. For example, prior to 1990, NASA's Jet Propulsion Laboratory launched (on average) five spacecraft a year, all of which were built internally. In the 1990s, JPL launched dozens of craft, most of which were built by external contractors. This policy led to certain high-profile errors which were attributed to production haste, poor communications, and mistakes in engineering management [31]. Accordingly, NASA terminated "faster, better, cheaper" and changed development methods.
- In 2003, the loss of the Columbia orbiter during re-entry prompted another major agency reorganization. Development budgets were significantly altered to fund the "return to flight" program that led to the successful relaunch, in 2005, of the Space Shuttle program.
- Currently, NASA is significantly reorganizing (again) around the new "vision for space exploration".

NASA's numerous national reorganizations have been reflected in the IV&V program. Figure 7 shows that, four times, the oversight for IV&V has passed between various NASA centers and NASA headquarters. Also, in 1996, NASA developed new contractual methods that allowed IV&V to be applied to a wide range of projects. These new methods resulted in a large change to IV&V management practices as the

[4] http://sas.ivv.nasa.gov/conclusions/index.php.

[5] http://sarpresults.ivv.nasa.gov.

year	#IV&V project at Fairmont IV&V	notes	oversight
1988	n/a	Space shuttle begins IV&V	Johnson Space Flight Center (Texas)
1991	n/a	Congress pass bill creating the IV&V facility in Fairmont, West Virginia	NASA headquarters (east coast)
1994	1	International space station IV&V at Fairmont begins	
1995	1		
1996	2		NASA Ames (west coast)
1996	3	New contracts enables IV&V for all NASA projects (but projects must fund IV&V from their own line items)	
1997	3		
1998	3		
1999	12	NASA mandates that IV&V be considered for all NASA software	
2000	15	Facility oversight moves to Goddard Space Flight Center	GSFC (east coast)
2001	20		
2002	36		
2003	42	IV&V now funded from a central agency source.	
2004	37		
2005	24	SILAP data collection begins	
2006	26		
2007	24		

Fig. 7 History of NASA IV&V

focus expanded from a few ongoing projects to dozens of smaller projects.

In 1999, after over a decade of success, IV&V was mandated to be considered for all NASA software. But a problem remained: projects had to find funds for IV&V from within their own budgets. This often slowed IV&V work since managers were forced to juggle the demands of development with the demands of IV&V. It took four more years of successful IV&V before NASA could solve this problem. In 2003, NASA reaffirmed and strengthened its support for IV&V by establishing IV&V as an agency-level program funded by the agency rather than by each individual project separately.

The above changes meant that more projects could receive the benefits of IV&V. However, the same changes that benefited NASA also complicated the twin goals of data collection and analysis. In nearly all projects, the schema of IV&V's issue tracking systems was modified to better address some specific requirement. While some data fields were required, most were not suitable for comparing the cost-effectiveness of IV&V across projects. Where possible, database schemas are standardized across multiple projects and multiple years (e.g., SILAP data has been collected since 2005). However, no data is collected consistently across all projects and all years.

Consequently, this study had to make do with the available data, none of which was collected especially for this purpose. Hence, it contains much noise (spurious signals not necessarily connected to the target of predicting issue frequency and severity).

4.2 Handling noisy data

When learning from noisy sources, it is important to use methods that aggressively prune noisy signals such as:

- The RIPPER [32] rule learner
- The WRAPPER algorithm [33], which explores subsets of the available features

Both methods are standard techniques for removing noise and generating succinct, easily explainable, models from data. RIPPER takes a particularly aggressive approach to pruning away superfluous parts of a model:

- After building a *rule*, RIPPER performs a back-select to see what rule conditions can be deleted, without degrading the performance of the rule.
- Similarly, after building a *set of rules*, RIPPER performs a back-select to see what *rules* can be deleted, without degrading the performance of the rule set. The learned rules are built while minimizing their *description length*. This is an information-theoretic measure computed from the size of the learned rules, as well as its errors. If a rule set is overfitted, the error rate increases, the description length grows, and RIPPER applies rule set pruning.
- RIPPER also employs a novel *rule set optimization* algorithm that tries replacing rules with some straw-man alternatives (i.e., rules grown very quickly).

RIPPER is one of the fastest rule learners currently known in the literature. In its haste, it may miss parts of the model that can be pruned. Hence, a much slower and more thorough feature pruner was also employed. Kohavi and Johns' RAPPER algorithm [33] explores subsets of the available features. Each subset is assessed by asking some target learner (in our case, RIPPER) to build a model using just that subset. The WRAPPER grows the feature set until the target learner reports that larger sets do no better than subsets. The algorithm stops when there are no variables left, or there has been no significant improvement in after the last five

1: Prevent the accomplishment of an essential capability; or jeopardize safety, security, or other requirement designated critical.
2: Adversely affect the accomplishment of an essential capability (no work-around solution is known); or adversely affect technical, cost or schedule risks to the project or life cycle support of the system, and no work-around solution is known.
3: Adversely affect the accomplishment of an essential capability but a work-around solution is known; or adversely affect technical, cost, or schedule risks to the project or life cycle support of the system, but a work-around solution is known.
4: Results in user/operator inconvenience but does not affect a required operational or mission essential capability; or results in inconvenience for development or maintenance personnel, but does not affect the accomplishment of these responsibilities.
5: Any other issues.

Fig. 8 Severities for robotic missions

1: A failure which could result in the loss of the human-rated system, the loss of flight or ground personnel, or a permanently disabling personnel injury.
1N: A severity 1 issue where a mission procedure precludes any operational scenario in which the problem might occur, or the number of detectable failures necessary to result in the problem exceeds requirements.
2: A cause loss of critical mission support capability.
2N: A severity 2 issue, where an established mission procedure precludes any operational scenario in which the problem might occur or the number of detectable failures necessary to result in the problem exceeds requirements.
3: A failure which is perceivable by an operator and is neither Severity 1 nor 2.
4: A failure which is not perceivable by an operator and is neither Severity 1 nor 2.
5: A problem which is not a failure but needs to be corrected such as standards violations or maintenance issues.

Fig. 9 Severities for human-rated missions

additions (in which case, those last five additions are deleted). Technically, this is a hill-climbing forward-select with a stale value of 5. Empirically, WRAPPER is known to be slower than other feature pruners, but yields best results [34].

4.3 Modeling the dependent feature

RIPPER inputs feature vectors, i.e., sets of independent features plus one dependent feature called the *class*. The goal of the learning is to find some combination of the independent features that predict for the dependent class feature.

In this study, the dependent feature was the number of issues found in a component, weighted by the severity. NASA now uses a standard five-point scale to score issue severity. A slightly different scale is used for robotic and human-rated missions (see Figs. 8 and 9) but the interpretation of the scales is same across all missions: severity 1 issues imply total disaster while severity five issues are ignorable.

For the purposes of analysis, the severity counts were summarized as follows. Let $s_{i,j}$ be the frequency of severity issue i in component j. If Max_i is the maximum number of issues of severity i, totaled across all components, then W_j is a weighted sum of all the severities.

The modeling intent of $W1_j$ was that any higher-level severity report is more important than any number of lower-level severities. In this interpretation, issue reports are *thresholds* that must be reached in order to *trigger* different activities. To implement this, the severity counts $s_{i,j}$ is divided by the maximum count for severity i:

$$W1_j = 1 + \sum_{i=1}^{5} \left(\frac{s_{i,j}}{1 + \mathrm{Max}_i} \right) \times 10^{6-i} \qquad (1)$$

(The “plus one” parts of these equations were added to avoid numeric errors. In this data, the severity frequency counts were large enough to make the “plus one” contributions negligible.)

4.4 Modeling the independent features

A learned model predicts for the dependent feature using some combination of independent features. The independent features used in this study came from NASA’s SILAP model [9,10].

Since 2005, NASA IV&V has been collecting a standard set of project features at the start of every IV&V project. SILAP uses these features to predicts for error potential (also known as likelihood) and consequence of error (also known as criticality).

SILAP was built via Delphi-sessions amongst NASA civil servants experienced with IV&V. This model holds the group’s consensus belief on what affects the quality of NASA projects. Figure 2 showed the SILAP derived factors that are computed from the raw factors using the functions of Fig. 10. These computed error potentials and consequences were then used to select IV&V tasks, using Fig. 3.

SILAP grew into an early life cycle project assessment methodology for IV&V. NASA civil servants report that SILAP has greatly clarified their discussions with projects regarding what IV&V tasks are/are not to be performed. Since SILAP was built locally at IV&V, it is understood

```
function CO( tmp)
        tmp=0.35*AS + 0.65 *PF;
        return (round((HS) > tmp) ? HS : tmp)
function EP() table
        return round(0.579*DV() + 0.249*PR() + 0.172*SC())
function SC()
        return 0.547*CX + 0.351*DI + 0.102*SS
function DV()
        return 0.828*EX + 0.172*DO
function PR()
        return 0.226*RA + 0.242*AM + formality()
function formality()
        return 0.0955*US + 0.0962*UC + 0.0764*CL  +
               0.1119*FR + 0.0873*DT + 0.0647*RM
```

Fig. 10 SILAP.awk: computes the derived features from the raw features of Fig. 2

goal	feature	weight	filter	weight* filter	contribution to goal
consequence	hs	0.35	1	0.350	35%
(co)	pf	0.65	1	0.650	65%
error	ex	0.828	0.579	0.479	47%
potential	cx	0.547	0.172	0.094	9%
(ep)	do	0.172	0.579	0.100	9%
	di	0.351	0.172	0.060	6%
	am	0.242	0.249	0.060	6%
	ra	0.226	0.249	0.056	5%
	us	0.0955	0.249	0.024	2%
	uc	0.0962	0.249	0.024	2%
	fr	0.119	0.249	0.030	2%
	dt	0.0873	0.249	0.022	2%
	ss	0.102	0.172	0.018	1%
	cl	0.0764	0.249	0.019	1%
	rm	0.0647	0.249	0.016	1%

Fig. 11 Relative contribution of Fig. 10 factors to the goals of consequence and error potential. The *percentages* shown on the *right* are normalized values for the weight × filter column

locally. Previous risk models, which had been developed elsewhere, were to a degree black-box models that were poorly understood. Hence, it was harder to defend their conclusions to clients lobbying for less IV&V oversight.

Each raw SILAP factor is weighted and, usually, filtered by a second numeric to derive the consequence and error potential values. The *weight* × *filter* column of Fig. 11 shows the relative contribution of each raw factor, as found by the Delphi sessions amongst the NASA civil servants.

This study could access SILAP descriptions of 280 components from 11 NASA projects. All these projects received IV&V in the period 2005–2007. Of these, only six of those projects had an issue tracking data schema that could generate a consistent set of issue data. Therefore this study was based on data from 211 components taken from six projects. None of these projects were human-rated missions. It was hoped that more data would be available but, given the pace of change within NASA databases, this was not the case.

Each component was described using:

- the 16 raw SILAP features of Fig. 2
- five issue severity counts $s_1, \ldots, s_5$
- the project name: $L1$
- the component type: $L2$; i.e., GNC (guidance, navigation, and control); PAYLOAD (specialized payloads); CDH (command & data handling), and GROUND (ground systems).

Note that the relationship of projects to components is one to many; i.e., the 211 components exist in one, and only one, of the six projects. Also, the relationship of components to component types is many to one, i.e., the 211 components are each assigned one, and only one, of the four component types.

4.5 Data quirks

Before applying automatic data mining, it is good practice to inspect the data distributions for any obvious anomalies. In the independent data, for example, all the data ranges over $1 \le x \le 5$ and there were no instances with $uc \in \{4, 5\}$ or $us \in \{4\}$ or $am \in \{4, 5\}$. Also, there were only $\frac{4}{211}$ examples of $ra \in \{2\}$. We can use this knowledge to rewrite rule conditions like $uc \le 1 \wedge ra < 3 \wedge am \ge 3$ into a more *simplified form* such as $uc = 1 \wedge ra = 1 \wedge am = 3$.

Also, in the dependent data, there are very few reports of severity 1 issues. In that project, the managers of that system were highly concerned about quality. Hence, they employed numerous best practices including high capability maturity model (CMM) practices (cl), extensive use of standards (us), and hiring very experienced developers (ex). That is, these practices did not *cause* the severity 1 issues in that project. Rather, practices were adopted *in response* to perceived issues with the project.

5 Learning SILAP2

5.1 Preprocessing

RIPPER's rules predict for discrete classes so, prior to learning, the Eq. 1 results for the 211 components were sorted and divided into five classes ("_1, _2, _3, _4, _5") with (roughly) equal population size (equal frequency discretization [35]); see Fig. 12. The quirks in the severity 1 data (discussed above) led to the following modeling decision: fuse together the top two classes, i.e., class "_12 = _1 ∪ _2".

5.2 Feature pruning

Figure 13 shows the WRAPPER results from ten experiments using RIPPER as the target learner. Each run used a randomly

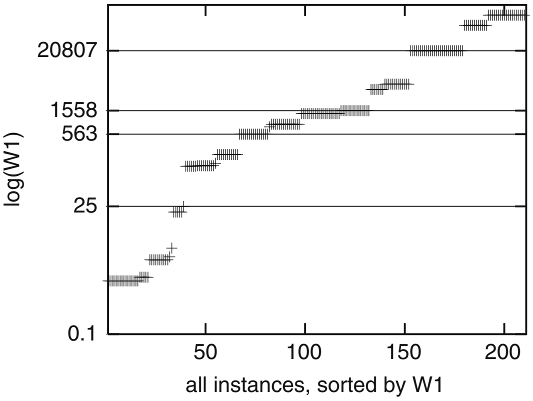

Fig. 12 Sorted values for Eq. 1 seen in 211 NASA software components. The y-axis shows the breaks b_1, b_2, b_3, b_4, b_5 that divide the data into five classes. Each class x comprises the data $b_x < x \le b_{x+1}$ or, for the top class, $b_5 < x \le \max$

group	feature	notes	number of times selected
1	us	use of standards	10
2	uc	config management	9
	ra	reuse approach	9
	am	artifact maturity	9
3	fr	formal reviews	8
	ex	experience	8
4	ss	size of system	7
5	rm	risk management	6
6	cl	CMM level	5
	dt	defect tracking	5
	do	development organization	4
	di	degree of innovation	4
	hs	human safety	3
	as	asset safety	2
	cx	complexity	2
	pf	performance	1

Fig. 13 Number of times a feature was selected in ten-way experiments where features were assessed using 90% of the data (selected at random)

selected 90% sample of the data. This figure is divided into *groups*:

- Group 1 are those features that were *always* selected in all ten pruning experiments. This group contains only one feature: use of standards (us).
- Groups 2 and 3 contain features that were *often* selected.
- Groups 4 and 5 contain features that were selected in the majority of experiments.
- Group 6 are those features that were *not* selected in the majority of the pruning experiments.

Group 6 raises some questions about the current version of SILAP. For example, the top five features of SILAP (measured in terms of *contribution to goal*, see Fig. 11) are hs, ex, pf, cx, and do. All but one of these (ex), appear in Group 6 of Fig. 13; i.e., are usually *not* selected by the WRAPPER. That is, empirically, these features are far less influential than suggested by SILAP. This result motivated us to develop a second version of SILAP.

To learn SILAP2, experiments were then conducted with RIPPER, using various feature subsets:

- The subsets of selected features found in the groups of Fig. 13;
- All the *raw* SILAP features;
- The project name ($L1$) or the component type ($L2$);
- Combinations of the top-most derived SILAP model: co, ep, and $co \times ep$.

5.3 Performance measures

The learned rules were assessed using F; i.e., the average f-measure seen in all the target classes. F is defined as follows. Let $\{A_x, B_x, C_x, D_x\}$ denote the true negatives, false negatives, false positives, and true positives, respectively, for class x. Recall (or pd) is:

$$pd_x = \text{recall}_x = D_x / (B_x + D_x)\,. \tag{2}$$

Also precision (or prec) is:

$$\text{prec}_x = \text{precision}_x = D_x / (D_x + C_x)\,. \tag{3}$$

The f-measure is the harmonic mean of precision and recall. If *either* precision or recall is low, then the f-measure is decreased. The f-measure is useful for dual assessments that include *both* precision and recall.

$$f_x = (2 \cdot \text{prec}_x \cdot pd_x)/(\text{prec}_x + pd_x). \tag{4}$$

Given n target classes, then one measure of the total performance is F, i.e., the average f_x measure seen in all classes:

$$F = \frac{1}{n} \sum f_x; \tag{5}$$

F ranges over $0 \leq F \leq 1$, and *larger* F values are *better*.

The F measures were collected in ten-way cross-validation experiments. To conduct such an experiment, the available data is divided into N buckets. For each bucket in a $N = 10$-way cross-validation, a model is learned on nine of the buckets, then tested on the remaining bucket. Cross-validation is the preferred method when testing predictors intended for predicting future events [8].

5.4 Results

Figure 14 sorts the F values found in ten-way cross-validation experiments on various feature subsets using RIPPER. The highest performing predictors used just $\frac{8}{16}$ of the SILAP features (see Treatment A). Also, nearly equivalent performance was found using just $\frac{4}{16}$ of the features (see treatment E).

One interesting aspect of Fig. 14 is that the derived features from the current version of SILAP (Treatment F) performed relatively poorly. This result is evidence that the current version of SILAP needs modification.

Also, using just the project name ($L1$) performed poorly. If this were otherwise, a project's name would be a good predictor for issue severity. It this were generally the case, then there would be no generality in NASA's issue reports.

Further, the component type ($L2$) proved to be a poor predictor of issue severity. That is, there was no evidence from this data that any of GNC, GROUND, CDH, or PAYLOAD are inherently more risky that any of the others.

Treatment E is very simple (using just four features) and performs nearly as well as anything else. Hence, the rest of this article will focus on treatment E.

The rules learned by treatment E using 100% of the training data are shown in Fig. 15. SILAP2 replaces the error potential calculations of Fig. 10 with Fig. 15. The issue

treatment	features	#features	_12	_3	_4	_5	$F = (\sum f) / 4$	
			f-measures (from Equation 5)					
A	all - L1 - L2 - group(6)	8	0.97	0.95	0.97	0.99	0.97	
B	all - L1 - L2 - group(5 + 6)	7	0.95	0.94	0.97	0.96	0.96	
C	all - L1 - L2 - group(4 + 5 + 6)	6	0.93	0.95	0.98	0.93	0.95	
D	all - L1 - L2	16	0.94	0.94	0.93	0.96	0.94	
E	all - L1 - L2 - group(3 + 4 + 5 + 6)	4	0.93	0.97	0.90	0.87	0.92	⇒ see Figure 15
F	{co*ep, co, ep}	3	0.94	0.84	0.55	0.70	0.76	
G	L1	1	0.67	0.69	0.00	0.46	0.45	
H	just îus”	1	0.64	0.60	0.00	0.00	0.31	
I	L2	1	0.57	0.00	0.32	0.00	0.22	

Fig. 14 Results from treatments. For a definition of the groups used in the second column, see Fig. 13

rule 1	if	$uc \geq 2 \wedge us = 1$	then	_5
rule 2	else if	$am = 3$	then	_5
rule 3	else if	$uc \geq 2 \wedge am = 1 \wedge us \leq 2$	then	_5
rule 4	else if	$am = 1 \wedge us = 2$	then	_4
rule 5	else if	$us = 3 \wedge ra \geq 4$	then	_4
rule 6	else if	$us = 1$	then	_3
rule 7	else if	$ra = 3$	then	_3
rule 8	else if	true	then	_12

Fig. 15 The eight rules found by treatment E

sort order: best=1 worst=27	*p*=phase	task	cost	frequency	frequency*cost
1	3. requirements	3.5	0.77	5	4
2		3.4	1.62	50	81
3		3.3	1.6	53	85
4		3.1	1.46	90	131
5		3.2	3.89	54	210
				sub-total	$\alpha_3 = 512$
6	2. concept	2.5	0.15	13	2
7		2.6	0.36	13	5
8		2.4	0.86	13	11
9		2.2	1.28	50	63
10		2.3	1.67	54	90
11		2.1	1.53	71	109
				sub-total	$\alpha_2 = 281$
12	4. design	4.5	1.43	5	7
13		4.6	0.63	35	22
14		4.3	2.50	9	23
15		4.2	4.31	31	134
16		4.1	2.71	81	220
17		4.4	2.84	88	250
				sub-total	$\alpha_4 = 655$
18	5. implement	5.11	1.82	3	5
19		5.4	1.16	26	30
20		5.5	2.34	26	61
21		5.3	5.06	53	268
22		5.1	3.49	81	283
23		5.2	10.28	53	545
				sub-total	$\alpha_5 = 1192$
24	6. test	6.4	2.08	26	54
25		6.3	20.93	3	63
26		6.5	3.20	26	83
27		6.1	2.30	88	202
				sub-total	$\alpha_6 = 402$

Fig. 16 Sort order of IV&V tasks: *top* to *bottom*, best to worst. Tasks are only listed if *both* “cost to apply task” *and* “frequency of applying that task” data is available. Costs frequency data is based on historical records at NASA IV&V. For proprietary reasons, cost numbers are normalized to 100. α_p is the sum of the frequency × cost figures for each phase

severity classes _5, _4, _3, _12 can be mapped to error potential 1, 2, 3, (4 or 5) of Fig. 3, respectively.

SILAP2 does not change the *consequence* calculation of Fig. 10. Human safety (hs) should always be more important that asset safety (as) or performance issues (pf) that compromise system goals. Otherwise, *consequence* can be modeled as a linear combination of as and pf.

6 Task pruning

Figure 3 can use SILAP2's calculation of *error potential* and SILAP's calculation of *consequence* to *select* a list of IV&V tasks. In two situations, that list should be *pruned*:

- If the tasks costs more than the available budget.
- If SILAP2's combining of _1 and _2 into _12 means that too many tasks are selected.

This last situation can occur with eight tasks of Fig. 3. With tasks 2.4, 2.5, 2.6, 4.7, 5.6, 5.8, 5.10, and 6.1, the tasks selected by *error potential* = 5 would not be selected when *error potential* = 4. This is not an ideal situation but, our current data has too few examples of severity 1 issues to support the generation of good level _1 predictors.

To support pruning, Fig. 16 sorts the IV&V tasks according to *phase effectiveness*, then *effort expended*. These sorts are explained below. Note that, if pruning is required, tasks lower in the sort should be pruned first.

Phase effectiveness was calculated using data from the nine robotic missions of Fig. 4. The data from the human-rated missions was ignored since launching humans into space is a highly specialized activity that may not be relevant to the broader field of software engineering.

An ideal set of IV&V tasks finds many issues, early in the life cycle, at low cost, after performing just a few tasks. This could be modeled as:

$$\gamma(\Delta) = \frac{\text{issues found early}}{\text{effort expended}} = \frac{\beta_p \cdot \Delta^{(6-p)}}{\alpha_p} \tag{6}$$

In this expression, γ is *phase effectiveness* and α_p (from Fig. 16) is total effort expended per phase. The β_p term (from Fig. 4) sums the issues found in phase p times a factor that weights high-severity issues exponentially more than low-severity issues. Finally, the Δ^{6-p} term (from Fig. 6)

p phase	$\gamma(\Delta = 2)$	$\gamma(\Delta = 5)$
2 concept	713 :18%	27,867 :39%
3 requirements	2,688 :67%	41,993 :58%
4 design	286 :7%	1,788 :2%
5 implement	182 :4%	454 :1%
6 test	146 :4%	146 :0%
totals	$\sum\gamma = 4{,}015$	$\sum\gamma = 72{,}248$

Fig. 17 Heuristic assessment of the relative merits of IV&V tasks at different phases. The α_p, $beta_p$ and Δ values come from and Figs. 16, 4, and 6

rewards issues found earlier in the life cycle at phase $p \in 2, 3, 4, 6$.

Figure 17 shows the γ calculated from our data. Each column shows the Eq. 6 values as a percentage of the totals for that column. The phases of Fig. 16 are sorted according to their γ values. To obtain a more fine-grained ordering, Fig. 16 sorts tasks within a phase by effort expended, i.e., α.

Note that, according to Fig. 16, most of the cost benefit of IV&V is seen in the early life cycle phases of requirements (first) and concept phase (second). Later life cycle IV&V tasks add very little to IV&V effectiveness (in Fig. 16, only 3–15%). This informs the problem of the eight cases where the rules of SILAP2 can overspecify the IV&V tasks. Note that five of those eight cases are late life cycle tasks, i.e. can be pruned without degrading much of the value of IV&V.

As to the remaining three problem cases (the concept phase tasks of 2.4, 2.5, 2.6), as shown by the "Z" notation in Fig. 3, these tasks are only for NASA-specific human-rated space missions. That is, software engineers using SILAP2 outside of NASA will rarely see task overspecification on tasks 2.4,2.5, and 2.6, and even if they do the net effect many be minimal. Figure 16 lists these tasks as having the smallest effort expended values of any of the concept phase IV&V tasks. Hence, if they are selected inappropriately, these tasks will add little to the total IV&V cost.

As an aside, we note that the sort ordering of phases in Fig. 16 is not without precedent. Writing over a decade ago, Higuera and Haimes [36] reviewed classes of risk associated with the Software Engineering Institute's risk taxonomy. They concluded that the risks associated with product engineering, development environments, and the general development program were roughly evenly distributed (30%, 33%, and 37%). However, Fig. 18 shows the risk within product engineering (i.e., the region usually studied by IV&V). Note that their risk distribution matches the general pattern of Fig. 17: late life cycle activities offer very little to the overall risk.

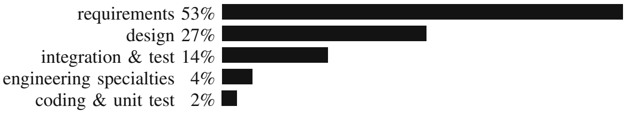

Fig. 18 Patterns of risk within product engineering; from [36]

6.1 Alternate sorting orders

In certain circumstances, the above pruning criteria should be changed. First, for mission- and safety-critical projects, pruning IV&V tasks is *not* recommended (to say the least).

Second, different organizations have different cost structures and, hence, different α_p values. If such data is available, then Fig. 16 could be sorted differently.

Third, the current version of SILAP offers some coarse-grained pruning heuristics that are different to Fig. 16. For example, to achieve *slight* pruning, all projects have a consequence *co* score of one *unless* $pf > 2 \vee as > 3$ (in which case, the *co* calculation of Fig. 10 is applied). SILAP2's pruning order (in Fig. 16) is more fine-grained than the pruning in the current version of SILAP.

7 External validity

7.1 Evaluation bias

The rules of SILAP2 were based on the issue/severity summation of $W1_j$ seen in Eq. 1. To check the external validity of that sum, we explored the effects of a very different calculation. Recall that, according to $W1_j$, issue reports are *thresholds* that must be reached in order to *trigger* different activities. An alternate interpretation, called $W2_j$, is that many lower-level severities can become more important than higher-level severities. In this interpretation, issue reports are *votes* that can *accumulate*. $W2_j$ can be implemented as follows:

$$W2_j = 1 + \sum_{i=1}^{5} s_{i,j} \times 10^{6-i} \tag{7}$$

The 211 $W2_j$ scores generated from our data are shown in Fig. 19. After dividing the scores into five approximately equal buckets, five new classes are generated: $\_1'$, $\_2'$, $\_3'$, $\_4'$, $\_5'$ representing least to most severe issues (respectively).

The *confusion matrix* of Fig. 20 compares the classes generated from Fig. 19 and those found in Fig. 12 (from Eq. 7): off-diagonal entries denote when the same class is assigned different classes by Eqs. 1 and 7. Note that, except for class _1 and _2, the classifications are nearly identical. Since the severity 1 issue data is quirky (see above), this study combined the classes _1 and _2 into their union $\_12 = \_1 \cup \_2$. Consequently, the region of confusion between Eqs. 1 and 7 disappears and the rules shown above are valid across a *range* of evaluation biases (from Eqs. 1 to 7).

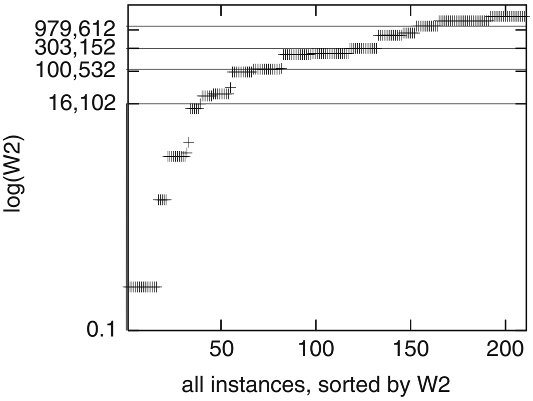

Fig. 19 Sorted values for Eq. 1 seen in the 211 NASA software components. The y-axis shows the breaks b_1, b_2, b_3, b_4, b_5 that divide the data into five classes

		W2				
		_1'	_2'	_3'	_4'	_5'
	_1	19	12			
	_2	27	20			
W1	_3			50	1	
	_4				42	
	_5					40

Fig. 20 Comparison of classifications offered by Eqs. 1 and 7 from Figs. 12 and 19. Each cell (x, y) counts how many components have class$(W1) = x$ and class$(W2) = y$

7.2 Language and search bias

All learners have a *language* and *search* bias. For example, linear regression often searches the Euclidean distance of actual to predicted values. The learned theory can only be reported in the language of a linear model, i.e., $y = K + \sum c_i x_i$. Other learners search information-theoretic measures of the data and their learned models are only reported in the language of decision trees or rules (e.g., RIPPER). Other learners have other search and language biases and will yield different models.

We must therefore defend the choice of learner used in this experiment. The case for RIPPER and WRAPPER was made above: our results are based on data not collected for the purposes of this study. Such data may contain signals not connected to our target function (Eq. 1). Hence, tools like RIPPER and WRAPPER that remove superfluous parts of the data or learned model are required.

As to other kinds of data miners, decision tree learners such as C4.5 [37] or CART [38] execute in local top-down search, with no memory between different branches. Hence, the same concept can be needlessly repeated many times within the tree. Consequently, such trees can be needlessly complex.

Also, linear regression may not be indicated in this domain since there is some evidence of nonlinearity in our data. Consider Fig. 15 and the situation where rules 1–6 have failed. In that case, the prediction falls to rules 7 and 8. If we change ra from 2 to 3 to 4, then the severity prediction changes in a nonlinear manner from _12, to _3, then back to _12 again. Such an effect cannot be modeled using the single linear model generated by standard linear regression.

Similarly, we would caution against learners that learn models containing a single feature; e.g., a study that reports that feature X has the strongest correlation to Eq. 1 and is therefore *the* most important indicator of issue severity. In all these experiments (Figs. 13 and 14) any single feature performed worse than those that used combinations of features. This is an encouraging result since, otherwise, experiments with data miners that explore combinations of features (such as RIPPER) would be superfluous.

There are more data mining methods besides RIPPER, WRAPPER, C4.5, CART, and linear regression. The improvement offered by those learners could only ever be $0.92 \leq F \leq 1$ (i.e., above the performance of treatment E). Hence, we are not motivated to move from RIPPER+WRAPPER.

7.3 Sampling bias

Apart from *language* and *search* bias, the results of any data mining experiment suffer from a *sampling* bias; i.e., the conclusions may be local to the data used in training and not be relevant to other populations. For example:

- All our conclusions were drawn from 211 NASA components and, as detailed above, that data has numerous quirks, i.e., missing ranges for $uc \in \{4, 5\}$, $us \in \{4\}$, $am \in \{4, 5\}$, $ra \in \{2\}$. SILAP2 should be used with caution in projects that fall into these missing ranges.
- This study reports that IV&V tasks performed in the concept phase are *less* useful than in the requirements phase. This result may be a result of current NASA development practices where more IV&V effort is devoted to requirements that concept analysis.
- The data used here comes from NASA and some of NASA works in a particularly unique market niche: human-rated missions to outer space. To minimize this concern, data from human-rated missions was not used to learn the rules of Fig. 15 or the sort order of Fig. 16.

Basili et al. [39] have argued that conclusions from NASA data are relevant to the general software engineering industry. NASA makes extensive use of contractors and these contractors service many other industries. Such contractors are contractually obliged (ISO 9001) to demonstrate their understanding and usage of current industrial best practices. For these reasons, we believe that the IV&V tasks proposed by SILAP2 would have general applicability across the industry. As evidence that this claim, we note that NASA did not invent Fig. 3 from scratch. Rather, it adapted it from IEEE standard V&V practices; one of us (K.S.) serves on the committee that defines and extends that standard [2].

8 Conclusions

Early detection of project issues can significantly reduce the cost-to-fix defects. Numerous prior studies have reported the effectiveness of such external IV&V teams at finding project issues early in the life cycle [1,4–7],

Nearly 100 (I)V&V tasks are known in the literature [1,2,9]. However, with respect to the cost-effectiveness of those tasks, the literature is incomplete. We can only access cost and effectiveness data on 27 of these tasks (see Fig. 16). Further, except in certain circumstances (see Sect. 6.1), we can only recommend 11 of those tasks for the purposes of IV&V. Perhaps it is time to spend less effort on the creation of *new* tasks and more effort on the assessment of *existing* tasks.

In order to assess the cost-effectiveness of IV&V tasks, we applied data miners to historical logs from NASA IV&V to find selection and ranking rules for IV&V tasks. Due to large-scale ongoing organizational changes within NASA, the data available to this study was potentially noisy; i.e., it contains signals not necessarily connected to the target of predicting for issue severity and frequency. Despite this, our data mining found rules (see Fig. 15) with very high precision and recall values (see Fig. 14) for predicting for different classes of issue frequency and severity (F values close to 1, in a ten-way cross-validation). This excellent performance persists across a range of summarization methods for issues and severities (from Eqs. 1 to 7). Other data miners *might* do better than on this data but that improvement could only ever be slight ($0.92 \leq F \leq 1$).

We attribute the success of the data mining to some careful choices regarding modeling and learning:

- WRAPPER [33] removes unnecessary or noisy features.
- RIPPER [32] also handles noise by aggressively pruning spurious parts of a learned model.
- High F values were achieved with WRAPPER+RIPPER *after* combining the severity 1 and 2 issue reports. This combination was necessary due to quirks in the relatively infrequent severity one issue reports in our data.

The RIPPER results comment on the merits of using linear models on these domain. The rules learned by RIPPER exhibit some nonlinearities; i.e., standard modeling methods such as linear regression would not be appropriate in this domain.

The WRAPPER results comment on SILAP, the current NASA IV&V task selection model. Comparing Figs. 11 and 13, it is clear that many of the features in the current version of SILAP are not useful for predicting issue and severity. Hence, a new version of SILAP was created. SILAP2 (in Fig. 1) uses the consequence calculations of the current version of SILAP, and replaces the old error potential calculations with the rules learned from WRAPPER+RIPPER. Since SILAP2 uses only a quarter of the SILAP, it is faster to apply.

When SILAP2 selects too many tasks, the selected tasks can be *pruned* (but note that IV&V task pruning is *not* recommend for mission- or safety-critical applications). The current version of SILAP has some coarse-grained pruning rules while SILAP2 offers a more detailed pruning criteria, using historical logs of expended effort and effectiveness (see Fig. 16).

The analysis of this report is based on public-domain data and software. To the best of our knowledge, this is the first *reproducible* report of a demonstrably useful issue frequency and severity predictor that can be quickly applied early in a project's life cycle. Reproducibility is an important pragmatic and methodological principle:

- Pragmatically, it allows industrial practitioners to tune our model to their local data regarding IV&V task cost-effectiveness (if such data was available).
- Methodologically, it allows researchers to confirm, refute, or even improve prior results.

In our view, in the field of software engineering, there are all too few examples of reproduced, and extended, results.[6] We hope that this report encourages more reproducible experiments in software engineering.

Finally, this analysis has used historical records of previously found issues to rank the IV&V tasks used to find them. A broader concern is what percentage of issues are missed by *all* IV&V tasks. Unfortunately, we have no data on this important concern. The goal of complete software assurance is illusive, e.g., the state-space explosion problem prevents model checkers from rigorously checking all but small mission-critical portions of a system [40]. The best we can do *currently* is to rank sets of candidate tasks according to their past cost-effectiveness and number of issues found. In the future, we can certainly do better than this, but only after corporations recognize the value in keeping consistent records, across multiple projects, across many years.

[6] Exception: see the reports of the PROMISE workshop http://promisedata.org/repository/papers.html.

- **am: Artifact Maturity**
  - 5: The project's development artifacts are more than 80% incomplete for the current stage of development based upon the project's selected software development life cycle.
  - 3: The project's development artifacts are less than 50% incomplete for the current stage of development based upon the project's selected software development life cycle.
  - 1: The project has the appropriate artifacts (less than 10% incomplete) for the current stage of development based upon the project's selected software development life cycle.
- **as: Asset Safety**
  - 5: complete loss of system.
  - 4: complete loss of primary subsystem.
  - 3: complete loss of other critical asset.
  - 2: partial loss of other critical asset.
  - 1: short-term partial loss of other critical asset.
- **cl: CMM level**
  1,2,3,4,5= CMM level 5,4,3,2,1. For a fast guide to assigning CMM levels, see the discussion on $pmat$ and "key process areas" in [41].
- **cx: Complexity**
  - 5: Complex scheduling, dynamic priorities, performance critical embedded system, complex parallelization, noisy stochastic data.
  - 3: Simple nesting, inter-module control using decision tables, message passing, middle-ware, simple i/0 processing (status checks and error processing), minimal structural changes to input files, functions behave differently in different models of system operation.
  - 1: Straight=line code, no nested structures, simple composition via procedure calls or some scripts, functions operate in only one mode, simple COTS-DB queries and updates.
- **di: Degree of Innovation**
  - 5: Basic principle being explored. Proof of concept exploration
  - 4: Component validation in lab or relevant environment.
  - 3: System prototype demonstration in relevant environment.
  - 2: System prototype demonstration
  - 1: Proven system concept via several successful prior projects.
- **do: Development Organization**
  - 5: Geographically disperse, multiple managers/ developers/ sub-contractors.
  - 4: As above, but one sub-contractor.
  - 3: As above, but no sub-contractors.
  - 2: All personnel co-located, sub-contractors used.
  - 1: All personnel co-located, no sub-contractors.
- **dt: Use of Defect Tracking**
  - 5: The project has no or a minimal defect tracking system.
  - 3: The project has a defect tracking system that is not consistent across the project. The defect tracking system is implemented in a manual means (e.g. a spreadsheet).
  - 1: The project has an integrated defect tracking system that allows defects to be assigned and corrected along with identifying root causes of the defects
- **ex: Experience**
  - 5,4: less than 5,10 years minimal domain or related experience (respectively).
  - 3: 10+ nominal domain or related experience.
  - 2: developed one like system.
  - 1: developed more than one like system or current incumbent.
- **fr: Use of Formal Reviews**
  - 5: The project has no planned formal reviews.
  - 3: The project has planned formal reviews, but no documented entrance and exit criteria.
  - 1: The project has planned formal reviews throughout the s/w life cycle that include entrance and exit criteria for the review
- **hs: Human Safety**
  - 5: Loss of life; i.e. a component receiving this rating is linked to an immediate and direct loss of life.
  - 3: Injury or potential for injury; a component receiving this rating has the potential to harm or injure a human. It can not directly be linked to a direct loss of life.
  - 1: Discomfort or nuisance: a component receiving this rating does not cause injury but may make a human uncomfortable (e.g. some aspects of environmental control) or be a nuisance.
- **pf: Performance**
  - 5: Unable to achieve minimum objectives/success criteria.
  - 4: Unable to achieve many objectives/success criteria.
  - 3: Unable to achieve a primary objective/success criteria.
  - 2: Unable to achieve a secondary objective/success criteria.
  - 1: None of the above
- **ra: Reuse Approach**
  - 5: Re-use approach is ad hoc and based on similarities between projects with no specific analysis of the item being re-used in terms of applicability to the current project.
  - 3: Re-use approach makes use of software artifacts that have been designed for other projects but not with re-use in mind. These artifacts are analyzed for applicability to the new development effort and the analysis is documented.
  - 1: Project has a documented re-use approach that is drawing from an establish re-use repository of software artifacts.
- **ss: Size of System** Measured in KSLOC. 1,2,3,4 = under 10K,50K, 400K, 850K. Otherwise, 5.
- **uc: Use of Configuration Management**
  - 5: Configuration management process is not documented/ not integrated into the software engineering process
  - 3: The project's approach to configuration management is not consistent across the project. Implementation is through manual methods.
  - 1: Project has a well-defined and documented process that maintains the integrity of the development artifacts through out the software life cycle through the use of a tried configuration management tool set.
- **us: Use of Standards**
  - 5: Ad hoc development, minimal documentation and planning and management.
  - 3: Using a documented standard and can demonstrate partial compliance to that standard.
  - 1: Integrated approach to sw/ development using a industry wide or locally proven process, documentation that explains modifications to that standard (if any). demonstration of compliance with the stated standard or modified standard.

Fig. 21 Ranges for SILAP and SILAP2 raw features

Appendix

The raw SILAP features are scored 5,4,3,2,1 representing what the SILAP authors [9] believed were the *worst* to *best* cases (respectively). Figure 21 offers some notes on how to select the right range for a particular component.

References

1. Wallace D, Fujii R (1989) Software verification and validation: an overview. IEEE Softw (May):10–17
2. IEEE-1012 (1998) IEEE standard 1012-2004 for software verification and validation
3. Boetticher R, Menzies T, Ostrand T (2007) The PROMISE repository of empirical software engineering data. http://promisedata.org/repository
4. Boehm B, Papaccio P (1988) Understanding and controlling software costs. IEEE Trans Softw Eng 14(10): 1462–1477
5. Dabney JB (2002–2004) Return on investment for IV&V, nASA funded study. Results available from http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp
6. Shull F, Basili B, Boehm B, Brown AW, Costa P, Lindvall M, Port D, Rus I, Tesoriero R, Zelkowitz M (2002) What we have learned about fighting defects. In: Proceedings of 8th international software metrics symposium, Ottawa, Canada, pp 249–258. Available from http://fc-md.umd.edu/fcmd/Papers/shull_defects.ps
7. Arthur J, Groner M, Hayhurst K, Holloway C (199) Evaluating the effectiveness of independent verification and validation. IEEE Comput (October):79–83
8. Witten IH, Frank E (2005) Data mining, 2nd edn. Morgan Kaufmann, Los Altos

9. Costello K (2005) Software integrity level assessment process (SILAP), NASA IV&V facility
10. Fisher M, Menzies T (2004) Learning iv&v strategies. In: HICSS'06, 2006. Available at http://menzies.us/pdf/06hicss.pdf
11. Jackson B, Griggs J, Costello K, Solomon D (2006) Systems level definition of iv&v. nASA document IVV 09-1, last revised March 16, 2006. Available online at http://www.nasa.gov/centers/ivv/pdf/170825main_IVV_09-1.pdf
12. Zelkowitz MV, Rus I (2001) Understanding IV&V in a safety critical and complex evolutionary environment: the nasa space shuttle program. In: ICSE '01: Proceedings of the 23rd international conference on software engineering. IEEE Computer Society, Washington, DC, pp 349–357
13. Easterbrook, S, Lutz RR, Covington R, Kelly J, Ampo Y, Hamilton D (1998) Experiences using lightweight formal methods for requirements modeling. IEEE Trans Softw Eng 4–14
14. Hayes JH, Dekhtyar A, Sundaram SK (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans Softw Eng 32(1):4–19. Available online at http://doi.ieeecomputersociety.org/10.1109/TSE.2006.3
15. Hayes J, Chemannoor I, Surisetty V, Andrews A (2005) Fault links: exploring the relationship between module and fault types. In: Proceedings of European dependable computing conference (EDCC), Budapest, Hungary, April. Available at http://selab.netlab.uky.edu/homepage/edcc2005_hayes_camera_ready_springer.pdf
16. Malin J, Throop D (2007) Basic concepts and distinctions for an aerospace ontology of functions, entities and problems. In IEEE Aerospace Conference, March
17. Lutz RR, Mikulski IC (2004) Empirical analysis of safety-critical anomalies during operations. IEEE Trans Softw Eng 30(3):172–180. Available online at http://csdl.computer.org/comp/trans/ts/2004/03/e0172abs.htm
18. Leveson N (1995) Safeware system safety and computers. Addison-Wesley, Reading
19. Heimdahl M, Leveson N (1996) NCompleteness and consistency analysis of state-based requirements. IEEE Trans Softw Eng (May)
20. Madachy R (1997) Heuristic risk assessment using cost factors. IEEE Softw 14(3): 51–59
21. Boehm B, Basili V (2001) Software defect reduction top 10 list. IEEE Softw (January):135–137
22. Menzies T, Stefano JSD (2003) How good is your blind spot sampling policy? In: 2004 IEEE conference on high assurance software engineering. Available at http://menzies.us/pdf/03blind.pdf
23. Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng (January). Available at http://menzies.us/pdf/06learnPredict.pdf
24. Menzies T, Chen Z, Hihn J, Lum K (2006) Selecting best practices for effort estimation. IEEE Trans Softw Eng (November). Available at http://menzies.us/pdf/06coseekmo.pdf
25. Glass R (1997) Software runaways: lessons learned from massive software project failures. Pearson Education
26. Raffo D, Nayak U, Setamanit S, Wakeland W (2004) Using software process simulation models to assess the impact of IV&V activities. In: Proceedings of the international workshop on software process simulation and modeling (ProSim'04), held in conjunction with the international conference of software engineering (ICSE), held in Edinburgh, Scotland, May
27. Jiang J, Klein G, Chen H, Lin L (2002) Reducing user-related risks during and prior to system development. Int J Proj Manage 20(7): 507–515
28. Ropponen J, Lyytinen K (2000) Components of software development risk: how to address them? a project manager survey. IEEE Trans Softw Eng (Feburary):98–112
29. Takagi Y, Mizuno O, Kikuno T (2005) An empirical approach to characterizing risky software projects based on logistic regression analysis. Empir Softw Eng 0(4): 495–515
30. Abe S, Mizuno O, Kikuno T, Kikuchi N, Hirayama M (2006) Estimation of project success using bayesian classifier. In ICSE 2006, pp 600–603
31. David L (2000) Nasa report: Too many failures with faster, better, cheaper, space.com, 13 March 2000. Available at http://www.space.com/businesstechnology/business/spear_report_000313.html
32. Cohen W (1995) Fast effective rule induction. In: ICML'95, pp 115–123. Available online at http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps
33. Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97(1–2):273–324. Available online at http://citeseer.nj.nec.com/kohavi96wrappers.html
34. Hall M, Holmes G (2003) Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans Knowl Data Eng 15(6):1437– 1447. Available at http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf
35. Dougherty J, Kohavi R, Sahami M (1995) Supervised and unsupervised discretization of continuous features. In: International conference on machine learning, pp 194–202. Available at http://www.cs.pdx.edu/~timm/dm/dougherty95supervised.pdf
36. Higuera R, Haimes Y (1996) Software risk management. Technical report, June cMU/SEI-96-TR-012
37. Quinlan R (1992) C4.5: programs for machine learning. Morgan Kaufman, Los Altos. ISBN: 1558602380
38. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Technical report. Wadsworth International, Monterey
39. Basili V, McGarry F, Pajerski R, Zelkowitz M (2002) Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory. In: Proceedings of the 24th international conference on software engineering (ICSE) 2002, Orlando, Florida. Available at http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/83.88.pdf
40. Menzies T, Cukic B (2002) How many tests are enough? In: Chang S (ed) Handbook of software engineering and knowledge engineering, vol II. Available at http://menzies.us/pdf/00ntests.pdf