2009 IEEE/ACM International Conference on Automated Software Engineering

Applications of Simulation and AI Search: Assessing the Relative Merits of Agile vs Traditional Software Development

Bryan Lemon, Aaron Riesbeck, Tim Menzies, Justin Price, Joseph D'Alessandro,
Rikard Carlsson, Tomi Prifiti, Fayola Peters, Hiuhua Lu, Dan Port*
Lane Department of Computer Science and Electrical Engineering, West Virginia University
*Information Technology Management, University of Hawaii
bryan@bryanlemon.com,tim@menzies.us,{tprifti|hlu3}@mix.wvu.edu
{aaron.riesbeck|justin.n.price|jdalessa57|ricca742|fayolapeters}@gmail.com,dport@hawaii.edu

***Abstract*—This paper augments Boehm-Turner's model of agile and plan-based software development augmented with an AI search algorithm. The AI search finds the key factors that predict for the success of agile or traditional plan-based software developments. According to our simulations and AI search algorithm: (1) in no case did agile methods perform worse than plan-based approaches; (2) in some cases, agile performed best. Hence, we recommend that the default development practice for organizations be an agile method.**

The simplicity of this style of analysis begs the question: why is so much time wasted on evidence-less debates on software process when a simple combination of simulation plus automatic search can mature the dialog much faster?

I. Introduction

There are too many examples in software engineering of positions being defended without empirical evidence. For example, Rumbaugh [1] and Jacobson [2] engaged in a high-profile and extended debate about the merits of different styles of object-oriented modeling- without citing any empirical evidence.

One explanation for this propensity to argue without evidence is the "data drought" reported by metrics-guru Norman Fenton. After years of advocating careful data collection [3], he now despairs of that approach. When data-based evidence is missing, model-based evidence can fill in the gaps. Sometimes, software process models are difficult to build and understand; especially when the uncertainties associated with the model lead to numerous wide ranges. Analysts may find the output confusing, rather than clarifying.

With the right automated support, *software process modeling can be very easy*. Given a well-defined theory and some automatic AI search tools, it is possible to collect model-based evidence about the value of variants on software processes.

Our case study concerns *requirements prioritization policies*. Such policies control the order in which a team implements the requirements. The spectrum of these policies can be characterized by two extremes: traditional *plan-based* and *agile*.

A standard argument is that agile should be used on highly dynamic projects, whereas plan based should be used on highly critical projects. Boehm and Turner [4] define five scales that can characterize the difference between plan-based and agile methods: project size, project criticality, requirements dynamism, personnel, and organizational culture. Previously, at ASE'08 [5], Port & Olkov & Menzies (hereafter, POM) studied the effects of different prioritization policies while adjusting the requirements dynamism. An advantage of model-based evidential reasoning is the ability to repeat prior analysis. For example, in the following study, the POM model is extended along the lines proposed by Grünbacher [6] then explored using an AI search engine.

Using the model and the AI search engine, we study plan-based and agile policies in different contexts. We found (1) no case where agile does worse than plan-based; (2) in some cases, agile performs much better. This leads to the conclusion: *The development methodology for an organization should be agile.*

II. Simulator Specifications

POM2 shuffles $requirements$ through three stages: "to do", "planned", and "finally completed". Within POM2 each $Requirement$ has a $cost$ and $value$. The requirements are organized in $trees$ which are assigned to $teams$. $Requirements$ are completed in $iterations$ according to the budget. The order $Requirements$ are completed is determined by the $requirements\ prioritization\ policy$. $Projects$ experience $early\ termination$ if funding is lost.

To simulate a $Project$, POM2 accepts 8 input values: $criticality$ (§II-D1), $crit.modifier$ (§II-D1), $culture$ (§II-D3), $initial\ known$ (§II-D2), $inter-dependency$ (§II-A), $size$ (§II-D4), $team\ size$ (§II-D4), and $dynamism$ (§II-D2).

These inputs effect the processing within POM2:

- In high $criticality$ systems, $requirements\ cost$ more to develop.
- In high $dynamism$, the $requirements$' $cost$ and $values$ change frequently.

1527-1366/09 $29.00 © 2009 IEEE
DOI 10.1109/ASE.2009.42

- Some organizational $cultures$ may frequently re-order $requirements$ while other organizational $cultures$ are more prone to ignore value change.

A. Requirements, Trees, Heaps, and Projects

Within POM2, a project is divided into teams, each of which implements a set of requirements with initial values and costs as defined by Port et al. [5]. These requirements are stored in acyclic trees representing the work breakdown structure. Each team is assigned a different tree from which new requirements are pulled. To model inter-team cooperation, at each level of the trees, one dependency is added between two trees at the same level. Depending on placement in the tree, inter-team dependency may significantly slow down the requirement completion process.

Trees are generated as follows. Nodes are assigned children according to an exponential distribution: $\frac{1}{X}\%$ of nodes have X children.

As a software project progresses it is not unusual for a development team to discover that they have new requirements to complete. We model this as follows:

- $number\ of\ requirements = size * 2.5$ [5]
- We pre-generate the number of new requirements to be added for each project iteration. Initially labeled as "hidden", the new requirements are marked "visible" when the simulator encounters the iteration for which they were added.

B. Iterations

Teams maintain a *plan*, which is a set of requirements that have been pulled from their heap. These requirements are then marked as known. *Plans* are processed in 2 to 6 iterations. Early terminations are very common in iterative projects if (for example) management decides that the resources of an organization should be transferred to a new project. After each iteration, the project has a probability of .9 to continue to the next iteration. This means that a project only has a 53% chance of finishing.

The number of requirements that can be completed in each iteration is controlled by the budget. Following POM [5], the budget is defined as:

$$budget = \frac{(total\ initial\ requirement\ cost\ )}{number\ of\ iterations} \quad (1)$$

C. Prioritization Policies

POM2 explores three prioritization policies.

1) Plan Based (PB) : A non-agile development method where requirements are sorted once at the start of development using the original $\frac{value}{cost}$ numbers assumed at the start of the project.
2) Agile2 (AG2) : A development method where requirements are sorted every iteration on policies sort requirements (every iteration) on $\frac{value}{cost}$.

D. Boehm and Turner Scales

POM2 implements four of the five scales identified by Boehm and Turner [7] that distinguished agile from traditional plan-based projects: project $criticality$, requirements $dynamism$, organization $culture$, and $size$. POM2 does not implement the fifth scale ($personnel$) due to theoretical reasons, see §II-D5.

1) Criticality: Boehm and Turner [7] measure criticality in terms of losses due to impact or defects and ranges from "none" (best for agile development) to "loss of many lives". Thus, plan-based methods are best suited to projects that must be carefully planned, lest defects cause loss of many lives.

We assume that X% of the teams are affected by the criticality, where $X = 2 \ldots 10$. Within our simulator, we refer to X as the criticality modifier. We adjust the cost of each requirement in a selected teams tree as follows:

$$cost' = cost * X^{criticality} \quad (2)$$

2) Dynamism: Project dynamism measures how frequently new requirements are created and how often existing requirements change value.

Boehm and Turner measure dynamism in terms of the percent of requirements changed each month and has the range 50% (best for agile) to 1 (best for planned-based).

To implement this factor, the POM1 assumptions for dynamism [5] are adopted:

- Initially, we only mark $30\% \leq 70\%$ of the requirements in the project tree as "visible." At each iteration, we make visible $new = Poisson(\lambda)$ more requirements.
- Another parameter controlling dynamism is σ. At each iteration, every requirement is visited, and and its value is altered by: $value'$ += $maxValue * N(0, \sigma)$
- After setting σ, we set λ to 10 % of σ.

3) Culture: For a fully agile project, changing requirement values means resorting the requirements in the plan to ensure that most cost-effective requirement is completed first. According to Boehm and Turner, culture is measured in terms of the percent of the staff thriving on chaos and has the range 90% (best for agile, flexible) to 10% (best for plan-based, rigid).

The calculation used for the accepted value is:

$$accepted = value + (value * N(0, \sigma) * culture) \quad (3)$$

4) Size: $Size$ is measured in terms of the number of personnel and has the range 3 (typically, best for agile), 10, 30, 100, and 300 (typically, best for plan-based). The size of a POM2 project is picked at random from this range and the number of requirements is then set to $size * 2.5$.

We planned to build teams using the results of [8] who report that the size of software development teams has $(min, mean, sd) = (1, 8, 20)$. However, this leads to a large number of single-person teams. We modified that result slightly in consultation with some of our NASA colleagues.

5) Personnel: Having described the scales implemented in POM2, we now discuss the theoretical problems that lead to an unimplemented $personnel$ scale. Boehm and Turner describe project $personnel$ using the Cockburn mixtures model. In summary, the mixtures model divides programmers into three groups:

Alpha: The most productive/flexible programmers.
Beta: Able to perform discretionary method steps.
Gamma: Unable/unwilling to follow shared methods

Applying the conventions of the Boehm and Turner *personnel scale*, lower personnel values indicate *more* alpha programmers on the team. Combining the COCOMO effort multipliers [9] and the Cockburn mixtures model, the net effect was nearly zero.[1] For that reason, POM2 ignores Personnel and will explore this issue in future work.

E. Performance Score

As POM2 requirements are completed, they are moved to a "done" set. The $Sum\ of\ Costs$ and $Sum\ of\ Values$ are plotted on a (X,Y) plane. When the project is finished, the plot can be compared to an *optimal frontier*, obtained by sorting all the "done" requirements using the final $\frac{value}{cost}$ for all those requirements. Note that no method can do better than the optimal frontier.

Scoring works as follows. If a project finishes after completing, say 100 requirements, then we would score the run as follows:

1 Take the final $\frac{value}{cost}$ curve for the project.
2 Divide that by the optimal frontier $\frac{value}{cost}$ curve.

III. Searching

Figure 1 is a distribution of 10,000 runs of POM2 scored as described above. Note that Agile methods maintain equal or better performance when compared to Plan-based methods. The Figure 1 results are interesting, but they are a two-dimensional summary of a ten-dimensional space. Before adopting the above conclusion, we need to find any special regions in the 10-D space of POM2.

We define these regions as follows: Given a model with $inputs$ to a model that are a set of $feature_i = range_i$, find the smallest subsets that most change the model $output$. Each such subset is special region since it is here where the score changes most. Our preferred tool for this analysis is KEYS [10]: a greedy search and a Bayesian ranking method called BORE. Assume that the model has N accessible states which are the inputs to the model (in our case, our sates are assignments to the Boehm and Turner scales). These inputs are randomized from a distribution $\{X_i, i = 1 \dots N\}$. A collection of inputs to the model is called a treatment.

$$\{x_1, x_2, \dots, x_N\} = treatment \quad (4)$$

[1]For a detailed explanation of the "zero effect" described, reference http://menzies.us/pdf/09pom2.pdf

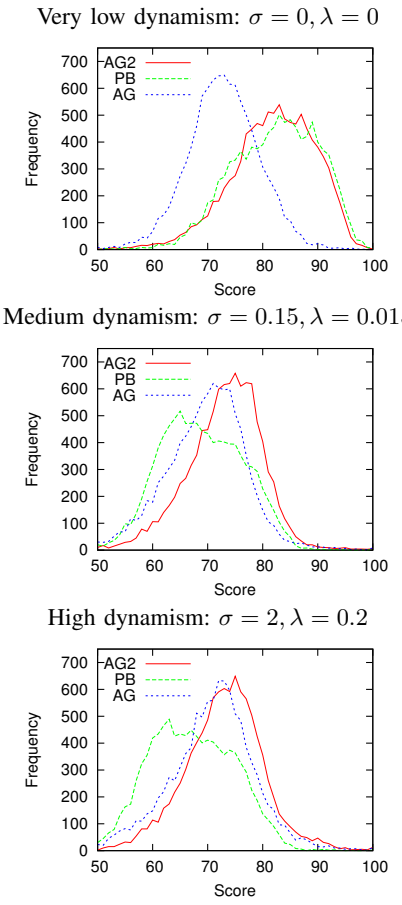


Figure 1: Distribution of Performance scores, controlling only σ and λ inputs,

After M number of initial samples, a second phase starts and one of the inputs is fixed to a desired range. This is performed for each of the input variables (the Boehm scales) until the stopping criteria is met. KEYS stops if the improvement from the previous round is less than 5%. The range is selected using the BORE heuristic. At the next run the $treatment = \{fixed\} \bigcup \{randomized\}$.

IV. Results

Figure 2 shows the ranges found in KEYS' treatments, while randomly selecting prioritization policies and the eight inputs on POM2. The results are split between two prioritization policies and three levels of requirements dynamism. Each combination of policy*dynamism was run 1000 times.

Notice that, in all cases, the treatments found by KEYS increased the performance scores reported in §II-E by a statistically significant amount (Mann-Whitney, 95%). Each cell in Figure 2 shows the percent frequency of that bin occurring in the treatments found by KEYS.

There are several majority-case effects in Figure 2 that deserve our attention. First, there are some strong negative results that hold across all dynamisms and prioritizations. One

		bin 1	bin 2	bin 3	bin 4	bin 5	bin 6	bin 7	bin 8	bin 9	bin 10
	criticality	.82	.86	.91	.95	1.0	1.04	1.08	1.13	1.17	1.22
	criticality modifier	2.0	2.8	3.6	4.4	5.2	6.0	6.8	7.6	8.4	9.2
	culture	0	10	20	30	40	50	60	70	80	90
	initial known	.40	.43	.46	.49	.52	.55	.58	.61	.64	.67
	inter-dependency	0	10	20	30	40	50	60	70	80	90
	size (total personnel)	3.0	32.7	62.4	92.1	121.8	151.5	181.2	210.9	240.6	270.3
policy / dynamism	team size (people per team)	1.0	5.1	9.2	13.3	17.4	21.5	25.6	29.7	33.8	37.9
plan-based / very low	criticality										
$\sigma = \lambda = 0$	criticality modifier										
	culture										
	initial known									2	4
	inter-dependency							1	1	1	
	size	89	2								
	team size										1
plan-based / medium	criticality								5	21	52
$\sigma = 0.15, \lambda = 0.0.15$	criticality modifier										
	culture										
	initial known										
	inter-dependency						1	2	6	6	8
	size										
	team size										1
plan-based / very-high	criticality							1	6	22	58
$\sigma = 2, \lambda = 0.2$	criticality modifier										
	culture										
	initial known										
	inter-dependency						2	3	6	10	11
	size										
	team size										1
agile 2 / very low	criticality										
$\sigma = \lambda = 0$	criticality modifier										
	culture										
	initial known							1	4	12	27
	interdependency						1	4	6	5	3
	size	72	1								
	team size									1	4
agile 2 / medium	criticality	1						1	1	1	1
$\sigma = 0.15, \lambda = 0.015$	criticality modifier							1	1	1	1
	culture						3	10	19	22	29
	initial known					1		2	2	2	2
	interdependency				1	1	1	1		1	
	size	97									
	team size		17	20	11	6	1				
agile 2 / very high	criticality	1				1		1	1	1	1
$\sigma = 2, \lambda = 0.2$	criticality modifier	1	1	1	1	1	1	1		1	1
	culture					1	4	13	17	20	22
	initial known						1	1	1	1	1
	interdependency		1	1	1	1	1	1	1		1
	size	100									
	team size		15	18	11	5	2	1			

Figure 2: The top table shows the division of each numeric inputs divided into ten bins. The rest of this figure shows the percent frequencies with which a range appears in the treatments found by 1000 runs of KEYS.

input ($criticality\ modifier$) was never selected by KEYS. Two inputs ($initial\ known$ and $inter-dependency$) were never selected in the majority case ($> 50\%$ of the runs).

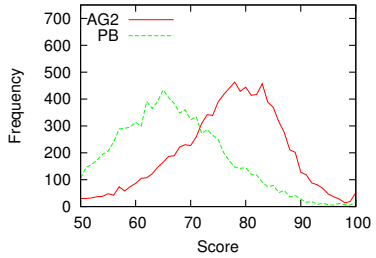
Second, with respect to agile2, there were frequent effects found for total $size$ of personnel, $team\ size$, and $culture$:

- In 72 to 100% of our agile2 runs, KEYS selected for the smallest possible total team size.
- In the upper bins for (medium, high dynamism), cultural factors appeared in (80,72)% respectively of the treatments found by KEYS.
- For medium and very high dynamism, it is useful to have team sizes of about 5 to 17 people (bin2, bin3, bin4, bin5).

Curiously, smallest size teams (below 5 people) was never selected by KEYS. We conclude that fails for very small teams when programmers spend the majority of their time interfacing with other teams.

Finally, Figure 2 shows that many results are very similar;

Medium dynamism: $\sigma = 0.15, \lambda = 0.015$



High dynamism: $\sigma = 2, \lambda = 0.2$

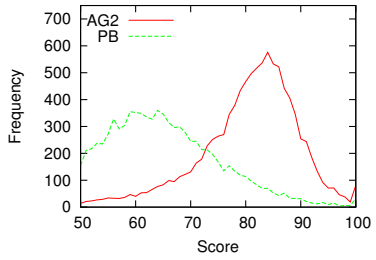


Figure 3: 1000 runs of KEYS controlling σ and λ while using $\frac{1}{10}$-th size ($3 \leq size < 32.17$) and top $\frac{1}{10}-th$ criticality (≥ 1.22) and choosing at random for the other inputs.

e.g. all the agile2 results offer nearly the same pattern. In fact, KEYS and Figure 2 show us that POM2 contains two major divisions of its 10-dimensional space. In terms of testing our general conclusion, the place to run tests is in the union of the two divisions: at very high criticality and very small overall team size. The results of this test is showed in Figure 3.

When POM2 was run in the union of these two divisions, agile2 produces much larger median scores than plan-based. This figure lends support to the general conclusion of this paper. If management is given a choice between agile2 and plan-based methods, they should adopt agile2.

V. Conclusion

Building software process models can be complicated by a drought of data. When data is scarce, model-based evidence can be used to make a reasoned case for reconfiguration of a project. To help mitigate information overdose, it is useful to augment a simulation engine with a search engine that can find the most interesting regions within the model input/output space.

This paper was a case study with combination of simulator (POM2) and AI search engine (KEYS). Our AI search engine explored the state space of our model to find regions that significantly improved the output performance score of the model. We found no case where plan-based out-perform agile2. In fact, sometimes agile2 performed much better than plan-based. We therefore recommend that:

The default development process be agile2.

Note also that in two areas, POM2 found limits in software engineering theory. In order to mature the conclusions of this paper, two issues need to be addressed:

- What is the effect of mixtures of different types of programmers on a project?
- What are the patterns of dependencies between requirements?

Very simple simulations/search engines can automatically explore the nuances of a debate. Using such automated tools, it is possible to make interesting conclusions with minimal machinery and effort. Our model definitions took just days of work; the implementation required a few weeks of work; and the runs to gather our data took just one night. Consequently, our conclusion is that evidence-less debates could be replaced, if possible, with debates that use model-based evidence. We concluded that agile2 performs as good or much better than plan-based development.

References

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson, *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[2] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[3] N. E. Fenton and S. Pfleeger, *Software Metrics: A Rigorous & Practical Approach (second edition)*. International Thompson Press, 1995.

[4] B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, no. 6, pp. 57–66, June 2003.

[5] D. Port, A. Olkov, and T. Menzies, "Using simulation to investigate requirements prioritization strategies," in *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, Sept. 2008, pp. 268–277.

[6] P. Grunbacker, "Personal communication," 2008.

[7] B. T. Boehm, "Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods," in *proceedings 26th International Conference on Software Engineering (ICSE)*, 2004, pp. 718–719.

[8] P. C. Pendharkar and J. A. Rodger, "An empirical study of the impact of team size on software development effort," *Inf. Technol. and Management*, vol. 8, no. 4, pp. 253–262, 2007.

[9] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[10] T. Menzies, O. Jalali, and M. Feather, "Optimizing requirements decisions with keys," *Proceedings PROMISE 08 ICSE*, 2008.