Using Simulation to Investigate Requirements Prioritization Strategies

Dan Port, Alexy Olkov
Information Technology Management
University of Hawaii, Manoa
Honolulu, HI
{dport, olkov}@hawaii.edu

Tim Menzies
Lane Department of Computer Science and Electrical EngineeringWest Virgina University
Morgantown, WV
tim@menzies.us

***Abstract*—Agile and traditional plan-based approaches to software system development both agree that prioritizing requirements is an essential activity. They differ in basic strategy - when to prioritize, to what degree, and how to guide implementation. As with many software engineering methods, verifying the benefit of following a particular approach is a challenge. Industry and student/classroom based experimental studies are generally impractical to use for large numbers of controlled experiments and benefits are difficult to measure directly. We use simulation to validate the fundamental, yet typically intangible benefits of requirements prioritization strategies. Our simulation is directly based on detailed empirical studies of agile and plan-based requirements management studies. Our simulation shows, as many have claimed, that an agile strategy excels when requirements are highly volatile, whereas a plan-based strategy excels when requirements are stable, and that there exist mixed strategies that are better than either for typical development efforts.**

Keywords-requirements, agile, plan-based, simulation

I. INTRODUCTION

All development efforts take great care in choosing what is implemented. A great deal of research and debate is directed on implementation approaches. In particular agile vs. plan-based [1,2] development approaches. Less attention is focused on what should be implemented when, yet this is no less important in today's complex and risky software development efforts. In this, prioritization of requirements is recognized as an essential micro-process within any development process [12]. With high customer expectations, tight schedules, and limited resources, prioritization is used to limit the scope [12] and deliver the most essential functionality as early as possible [14]. It is an accepted fact that for most development efforts that not all identified requirements will be implemented. Wiegers [14] states that prioritization is needed, not just so as to be able to ignore the least important requirements, but also to help the project manager to resolve conflicts, plan for staged deliveries, and make the necessary trade-offs throughout the development lifecycle [14].

Both plan-based and agile development approaches view prioritization as a fundamental activity [1,12] but they differ in their basic strategy. A requirements prioritization strategy determines what requirements are implemented and in what sequence with respect to a strategic goal such as "minimize cost." There are many different strategies. For example, "implement the lowest cost requirements first" or "implement the highest value requirements first" and some strategies are more effective than others.

The primary question of interest here is in finding an effective strategy for a given development effort. While there is a great deal of literature on requirements prioritization, little of this addresses the issue of strategy effectiveness. Perhaps one reason for this is that, with the exception of naïve strategies (e.g. implement the requirements as they appear) all strategies rely on difficult assessments such as cost estimation, value assessment, dependency analysis, and so forth. While estimating the cost of a task is generally straightforward, it is difficult to estimate the cost of a particular requirement (cost here typically is interpreted as "effort" here, not money). Value is generally an "intangible" not easily attributed to a particular requirement. Generally it is overall value, or the value for completed groups of requirements that represent a complete set of functionality is all that is considered. So called "earned value" is not actual value [3] is not reliable for prioritization purposes. Furthermore, requirements prioritization is difficult to monitor and measure "in-vitro" within actual practice [13].

Given the above issues, and many others that we have left out, the research question we are interested in is what is a practical means for investigating the effectiveness of requirements prioritization? Controlled experiments are impractical, as is common with assessing software engineering methods (e.g. how to set up exact replications with different strategies, how to prescribe requirements volatility, etc.). In addition to the above stated challenges in dealing with intangibles and collecting data in-vitro, experiments would require a large number data points to get convergence of effectiveness measures due to the highly variable (and uncontrollable) conditions and circumstances within any given project.

Comprehensive simulation is an attractive option for investigating and providing empirical support and justification of new software engineering methods whose effectiveness measures are intangible and unobservable. Such simulations are common and accepted as evidence

within the management and operations research literature where the evaluation challenges are analogous to those in software engineering. In this work we create a simulation based on requirements theory and a detailed empirical study of requirements practices. We verify that the simulation is consistent with the "home-ground" theory of Boehm-Turner [2] for basic agile and plan-based requirements prioritization methods. The simulation is then used to explore properties of requirements prioritization strategies and investigate two new methods suggested by application of this theory. Strategies are compared graphically and with respect to six strategy effectiveness measures under various requirements volatility scenarios.

The home-ground theory states that agile methods are most effective when requirements volatility is very high, while plan-based methods are most effective when there is relatively little requirements volatility. The theory suggests that a mixture of the two methods will generally be more effective than either alone for typical development efforts. This study seeks to answer the question "what would a mixed agile and plan-based requirements prioritization strategy look like and how effective is it?

The paper begins with an overview of how requirements are modeled in the simulation, their evolutionary stochastics, and the adjustable parameters of the simulation. This is followed by a detailed discussion on the assumptions made in the simulation model and their justifications. We then describe how the agile and plan-based prioritization strategies are modeled and simulated. Then a discussion on measuring the effectiveness of a strategy and describing six relevant effectiveness measures we use for generating comparison results. Subsequently we discuss the theoretical validity of the simulation model. This leads to the construction of two new "mixed" strategies which are then compared to the agile and plan-based methods in the results section. We conclude with a brief discussion of contribution of this work, the utility of simulating software development micro-processes such as requirements prioritization, and indicate how the new prioritization strategies can be applied in practice.

II. Simulating Requirements Evolution

For our purposes we are only interested in quantitative evaluation attributes of a requirement. The commonly used attributes for this are the *cost* of implementing (this is usually effort rather than a monetary unit) and the expected gain if the requirement is implemented which is referred to as *value*. A requirement R_i is considered an ordered pair ($cost_i$, $value_i$) where `min_cost` $\leq cost_i \leq$ `max_cost` and `max_value` $\leq value_i \leq$ `min_value`. A "base set" of requirements $\{R_1, R_2, \ldots, R_{num\_reqs}\}$ is generated by assigning uniform random variables $cost_i = \boldsymbol{U}$(`min_cost`, `max_cost`) and $value_i = \boldsymbol{U}$(`min_value`, `max_value`). After each iteration, requirements volatility is handled by updating each requirement value with a normally distributed random variable $R_i = (cost_i, value_i + \boldsymbol{N}(0,$ `req_value_sigma`$)$) and possibly a Poisson number of new requirements ***Poiss***(`ave_new_req_per_iter`) are added to the base set. Cost is assumed to be non-volatile for reasons explained in the assumptions section. If $value_i < 0$ then $value_i = 0$ from that point on (requirements do not rise from the dead after being completely devalued). A base number of iterations `num_iters` and a minimum number of iterations `min_iters` are chosen. The requirements change values every (*total cost of base reqs*) / `num_iters` which may not be the same as the development iteration size. Note that strategies do not have to use this as their iteration size determination. It is mainly for handling requirements volatility.

The simulation runs for at least `min_iters` iterations applying requirements prioritization strategies to the current set of known requirements. After `min_iters` iterations, a "stopping time" is determined by when a Bernoulli random variable ***B***(`end_dev_prob`) = 1, after which the development is considered ended (this simulates the unknown stopping time for a development project) and this completes one trial in the simulation.

TABLE I. Simulation Parameters

Parameter name	Description
`num_trials` (1000)	Number of simulation experiments to perform
`min_value` (30)	Lower bound on requirements values
`max_value` (500)	Upper bound on requirements values
`min_cost` (1)	Lower bound on requirements costs
`max_cost` (100)	Upper bound on requirements costs
`rank_tol` (10%)	Two goodness measures are considered the same rank when they are within this percentage of each other
`num_reqs` (25)	Number of initial base requirements
`num_iters` (6)	Number of initial base iterations (a maximum on the development)
`req_value_sigma` (15%)	Standard deviation of the requirements value volatility Normal random variable
`ave_new_req_per_iter` (1.4)	Average arrival rate of new requirements per iteration (Poisson)
`ave_unimpl_req_per_iter` (20% of `num_reqs`)	Average arrival rate of requirements taken from the initial base set (for "discovery" of initial requirements in later iterations by agile methods)
`inital_bound_lower` (30%)	Minimum percentage of initial base set requirements that can be "discovered" (for initial requirements set for agile)
`initial_bound_upper` (70%)	Maximum percentage of initial base set requirements that can be "discovered" (for initial requirements set for agile)
`min_iters` (`num_iters`/4)	Minimum number of iterations for a trial
`end_dev_prob` (1/`num_iters`^0.333)	Parameter for the Bernoulli random variable to determine a random development stopping time (compounds after `min_iters` is exceeded)

No strategy may exceed the total cost of the base requirements at the stopping time, but they may expend less due to the particular cost of the requirements left over at the

last iteration (i.e. if adding one more would exceed the stopping cost, then it is not used).

Table I provides a summary of the adjustable parameters for the simulation and the default parameters for a "typical" simulation (in parenthesis).

In our investigation we made use of the default parameters indicated in Table I to generate the results described in this paper. The selected values appear to be representative of "typical" development efforts, but we do not have explicit evidence to support this currently. We have performed a sensitivity analysis on the parameters and found that beyond using excessive values, our results are not sensitive in any of the parameters not directly related to requirements volatility. This suggests that regardless of what the actual parameters are for typical projects, we would not expect the results would be any different.

III. Assumptions of the Simulation Model

In this section we describe the significant assumptions used to define the simulation model and the justifications for using them.

End development time is unknown: even with exhaustive cost and schedule estimation and manage-to-plan development, high uncertainty and business circumstances frequently expand or contract expected development effort. According to the 2006 Standish CHAOS [15] survey, only 35% of development efforts are successful in meeting cost, schedule, or capability expectations. To simulate unknown end development "stopping time" we assume a minimum number of iterations are completed after which a biased coin toss is used to determine if the development must stop. No requirements may be implemented that exceed this stopping time. Without this assumption, all strategies would implement the complete set of identified requirements and thereby achieve the same total value and total cost. Some of the more interesting strategy effectiveness properties appear when only a subset of the requirements are implemented. When there is no requirements volatility, an unknown end development time generally results in some of the identified requirements not being implemented. In a volatile requirements situation, it is possible that the stopping time will exceed the time needed to implement the base requirements as new requirements may be added during the iterations.

Requirements volatility has two independent factors: requirements are assumed to change either by adding or removing a requirement, or by a change in a requirement value or cost. This is substantiated in [4]: "Participants reported two types of requirements changes: adding or dropping features, and changing already implemented features."

Non-volatile cost: it is assumed that requirements volatility affects value but not cost. Uncertainties in cost can cause volatility, however in our analysis we have found that this simply increases the overall volatility and does not constitute an independent volatility factor. Including cost volatility did not change our results, but it does complicate the analysis, so as a simplification we assume cost is non-volatile and this does not invalidate our results. Note that the cost of late changes to requirements may be a factor here not accounted for that perhaps should be investigated. Such cost of change factors are generally thought predominantly due to requirement dependencies which we have determined do not affect our results, hence we suspect that this too would not change our basic results .

Change in value due to volatility is normally distributed: there is no evidence to support that value changes are biased either in favor of higher or lower valuations as development progresses. Furthermore while minor changes of requirement value are common, major changes are rare. This is indicated in [4]: "Customers provide feedback and can request major changes if their expectations aren't met. In the 16 organizations [surveyed], this kind of change is relatively rare" and "… most of the change requests are 'usually more a case of tweaks…" Even though these quotes were made in the context of changes requested within an agile development effort, they indicate the general nature of requirements changes because it is not the agile process that dictates the magnitude of a change in requirements value. A random variable whose value varies either up or down equally (i.e. symmetric) and tends to vary relatively little from its average but on rare occasion varies greatly is modeled very well with the Normal distribution.

Once a requirement has zero or negative value it never regains positive value: there is no evidence to support that when a requirement is assessed as valueless that it ever is reconsidered and becomes valued. Requirement reconsideration after being de-valued, is more accurately modeled as a new requirement that was "inspired" by the de-valued requirement.

Arrival of new requirements due to volatility is Poisson distributed: there is no evidence to support that the number of new requirements in any given iteration is dependent of the number of new requirements previously added or removed. Note that this is NOT saying that new requirements are independent of previous or existing requirements. Only the number of "new arrivals" are independent. We further assume that new requirements are "orderly" and do not arrive simultaneously at any given time, although many can arrive within the same time interval (this is a minor technical point that we do not believe is relevant to requirements prioritization). Thus the number of new requirements arrival is a "memoryless" and "orderly" counting process modeled as a Poisson random variable.

Requirement implementation dependencies are not accounted for: it is generally thought that requirements are rarely independent of each other and that such dependencies must be accounted for when prioritizing. While such dependencies are critical for actual implementation, this is not the case for prioritization. Prioritization is an assessment activity that influences, but does not determine the implementation. We have found two important factors in this regard. First, when requirements are assessed for prioritization, dependencies tend to be implicitly accounted for – costs and values are adjusted to match dependencies

e.g. if requirement B depends on A which has a high value, then so does B. That is, it is not individual requirements being prioritized, rather sets of requirements (and perhaps partial requirements) that constitute single implementation activities. Second, prioritization is only a "first pass" plan that must be later refined into a practical implementation plan when the particular dependency issues are known. On this point, we have not seen any evidence that developers explicitly account for dependencies when "first pass" prioritizing requirements, while there is ample evidence that dependencies are considered for implementation (but surprisingly rarely considered explicitly). Given that a prioritization operates on the exhaustive space of all possible orderings and combinations of requirements, the effect of individual dependences does not appear to influence the net outcome. This fact has been verified in our simulation. We have conducted simulations accounting for varying degrees of dependencies (using so-called dependency graphs), both extreme and mild and have observed no substantial difference in simulation behavior. Given this and that accounting for dependencies greatly complicates the simulation, we have chosen to ignore them for studies of prioritization strategies. It is certainly possible that requirement dependencies do matter for some prioritization properties and we are not claiming otherwise here. For our study, requirement dependencies in general affect all strategies equally, and as such they do not affect the properties we are investigating, i.e. requirement dependencies are independent of prioritization strategy. As such, we are confident in ignoring such factors at this time.

Single option for implementation: generally there are many choices on how a requirement could be implemented; each with a different (*cost*, *value*) pair. While a prioritization strategy involves choosing how to implement, as well as the order in which to implement a requirement, this represents an independent dimension on the effectiveness of a strategy (a fact that was shown in [11]). For our investigation it is assumed that the optimal implementation option is always chosen and this option will achieve the entire value and expend the entire cost of the base requirement. We have extended the simulation to include implementation options, but this is not the main focus of the investigation described currently nor does it change any of the basic results presented here.

IV. 4. Simulating Prioritization Strategies

The prioritization strategies are modeled on empirical results from [3,4,5,7,9,10]. Our models implement a literal representation of agile and plan-based methods as they are defined in the literature. In [2] Boehm-Turner argue that the general perception, as well as is stated in the literature, that development practice follows one of two extremes – either agile or plan-based. Given that neither approach is "at home" for typical development efforts, Boehm-Turner claim that this view is unrealistic and leads to confusion and decreased effectiveness in practice. We state up front that the agile and plan-based strategies we simulate likely *do not represent* what is actually used in practice. Since we are interested in investigating properties of prioritization strategies, we must consider what is considered to be at the extreme. Without explicitly designing the simulation to do so, the results of our simulation support Boehm-Turner's assertion that, in general a "balance" of plan-based and agile methods is indeed the best approach.

The literal view of agile and plan-based approaches we take simply means that we do not mix (or balance in the terminology of [2]) the activities that are attributed to each respective approach. However, we most certainly desire that the activities performed in each strategy are representative of actual practice and not on specious statements, theories, or claims that have not been empirically validated or justified. We are fortunate to have at hand a comparative empirical study of requirements practices across 16 different companies [4] to provide a reliable basis for establishing a representative simulation of agile and plan-based requirements prioritization strategies. From this study the following properties of agile and plan-based strategies, with respective evidence, form the basis for simulating these approaches realistically. The quotations below are taken from [4] and indicate the rationalization for the stated properties.

Agile requirements prioritization properties

1) Requirements are prioritized at the beginning of each development iteration: "… agile involves prioritizing requirements in each development cycle. Prioritization often happens … at the beginning of each cycle."

2) Requirements are prioritized according to highest value first: "… agile practitioners uniformly reported that their prioritization is based predominantly on one factor – business value …"

3) A significant subset, but not all "initial" requirements are discovered at the first iteration: "… requirements aren't predefined; instead, they emerge during development. High-level RE occurs at the project's beginning. During this brief process, the development team acquires a high-level understanding of the application's critical features."

It is difficult to gauge exactly what percentage of the requirements are identified at the beginning of an agile based development. In the simulation we randomly (according to the uniform distribution) select a percentage from `initial_bound_lower` to `initial_bound_upper` of the base set of requirements (which are the assumed base of all "actual" requirements that could be discovered at the start of the project). There is little empirical study of what range might be representative and there are numerous conflicting practices here. All we know is that in general not all initial requirements are identified. Indeed, it is a primary tactic of the agile approach NOT to discover the majority of initial requirements and to allow requirements to emerge during the development as they become important. To be conservative we selected a range of 30%-70% for the agile approach.

With the above properties it is fairly straightforward to simulate an agile prioritization strategy. Agile iterations will be determined inductively by:

Iteration 0: define a uniform random integer `first_iteration_number` between `initial_bound_lower*num_reqs` and `initial_bound_upper*num_reqs` to determine the number of initial critical requirements identified from the base set. We define AG_PLAN as the working set of requirements for the agile strategy. At the end of each iteration this ordered set will represent the prioritization for that iteration. At the start of development this is the sub-set of the base requirements initially identified, say $\{R_1, R_2, \ldots, R_{\texttt{first\_iteration\_number}}\}$ (recall agile property 3 from above) and the remaining base set requirements are placed in AG_HEAP as $\{R_{\texttt{first\_iteration\_number}+1}, \ldots, R_{\texttt{num\_reqs}}\}$. We now define the iteration size. For simplicity we choose this to be the same as the iteration size used for volatility adjustments (however note this synchronization actually gives an agile strategy a bit of an unfair advantage in the final iteration as will be discussed later).

Iteration k>0: sort AG_PLAN from highest to lowest value. Implement the requirements in this set in this order up to the iteration size. Unimplemented requirements are carried over to the next iteration.

Intra-iterations: a Poission random variable with intensity `ave_unimpl_req_per_iter` number of requirements are taken from AG_HEAP and added to AG_PLAN to represent the "discovery" of the base set requirements as they emerge during the development.

Plan-based requirements prioritization properties

1) Requirements are prioritized exhaustively once at the beginning of the development: "… in traditional RE, requirements are typically prioritized once."

2) Requirements are prioritized according to highest cost-benefit first: "… in traditional RE, many factors drive requirements prioritization – for example, business value, risks, and cost ..."

Other empirical works within requirements engineering have described the predominance of cost and benefits as the primary prioritization decision factors [5,9,14]. It has been shown that the optimal strategic prioritization order is established by considering the value/cost (or slight variations of this) [11]. Requirements engineering methods are consistent with this, for example in the popular cost-value graph prioritization approach [6].

Iteration 0: Owing to property 1 above, the PB_PLAN is the entire base set of requirements as $\{R_1, R_2, \ldots, R_{num\_reqs}\}$. We sort this set from highest to lowest value/cost of the requirements. The iteration size is inconsequential to the plan-based strategy as no changes to the priorities are made subsequently.

Iteration k>0: Implement the requirements in PB_PLAN in the order established in iteration 0.

Intra-iterations: since requirements are prioritized only once, new requirements that emerge are "unexpected" and simply appended to the end of PB_PLAN in the order that they arrive.

V. Six Measures of Strategy Effectiveness

A strategy $\{R_1, R_2, \ldots, R_n\}$ can be visualized by plotting the cumulative values versus costs (i.e. running sums). That is, $(cost(R_1), value(R_1))$, $(cost(R_1)+cost(R_2), value(R_1)+value(R_2))$, … as exemplified in Figures 1-3. However there are several possible "best strategy" criterions that are of interest which may not be obvious from a visual inspection. To automate the analysis of strategies, we introduce the following six "goodness of strategy" metrics related to the strategy graphs as exemplified in Figures 1-3:

tv – total value of the requirements implemented. This represents the value on the y-axis at the end point of a strategy graph (or the ultimate "height").

tc – total cost of the requirements implemented. Requirement costs are assumed to be constant throughout the project. . This represents the value on the x-axis at the end point of a strategy graph (or the ultimate "length").

int – the discreet integral, or total area under the strategy curve from 0 to **tc.** This represents the "total value created" by a strategy (different than total value implemented).What makes this measure attractive is that it accounts for the "goodness" of a strategy overall rather than just at the end. That is, a strategy may perform poorly early on, but jump up at the end to deliver a good end value. But if the development had stopped earlier, it would have done poorly at the end. Another way to look at this is that the closer a strategy comes to the optimal strategy that could have been created has the unknown values and requirements been known (we call this the frontier strategy), which no strategy can do better, then the "better" that strategy. The frontier strategy has the obvious property that it has the maximum possible area under it, so the larger a strategies integral is, the closer it must be to the frontier strategy.

ben – is the benefit, defined as **tv – tc**

cb – is the cost-benefit defined as **tv/tc**

fr – it is the "frontier ratio" defined as **int/(int for frontier strategy up to same cost)** where the **frontier strategy** is the Pareto ordering of the set of all the requirements at the stopping time. No strategy can pass this curve and hence it represents the "optimal" in terms of overall value and cost. However note that this is an "after the fact" ordering that no strategy can generally achieve due to the random changes in values and new requirements added during the iterations. The closer this ratio is to 1, the more closely it resembles the frontier strategy up to its **tc**.

When calculating goodness of strategy metrics, only the values of the requirements at the stopping time are used as

these presumably represent the value that is actually delivered.

VI. Theoretical Validity

Our objective for simulation is to explore comparative properties of prioritization strategies under various conditions that are difficult or impossible to observe in practice. To inspire confidence in the validity of our simulation it is important that it is consistent with known theory. A well-accepted basic model for comparing plan-driven and agile methods is the "home ground" model of Boehm- Turner [2]. In this model, plan-based and agile methods are considered to have their "home-ground" at opposite extremes with respect to five project factors – Size, Criticality, Dynamism, Personnel, and Culture. The theory states that most projects will have some low factors, some high, and some in between and rarely wholly within either method's home ground. Thus using a mixture of approaches to best match the factor values is generally more effective for any given project. The most relevant factor for requirements prioritization is Dynamism, defined as the percentage of requirements-change per month. According to [2], agile methods have a home ground when there is 50% or more changes per month, while plan-based methods excel at 1% or less. In our simulation high Dynamism translates into large values of the parameters `ave_new_req_per_iter` (**λ** for convenience) and `req_value_sigma` (**σ** for convenience), the opposite for low Dynamism.

For our simulation, this aspect of the home-ground theory implies that an agile requirements prioritization (AG) should excel with (high **λ,** high **σ**), whereas plan-based (PB) excels with (low **λ,** low **σ**) as summarized in Table II below.

TABLE II. Dynamism and Simulation Parameters

average new requirements per iteration λ		requirements value standard deviation σ	
		LOW σ	HIGH σ
HIGH λ		*?*	*AG*
LOW λ		*PB*	*?*

Our simulation results are consistent with Table II with very high confidence. Specifically, there is a significant difference in the ranks of all goodness of strategy measures except **tc** with (high **λ,** high **σ**). In this case it is not unexpected that PB will have slightly better **tc** (which implies lower cost) given the last requirement implemented in this strategy is likely to have much lower cost than the requirement that would have been implemented had development not ended abruptly. We see on the strategy graphs that PB has greater "pull-back" at the stopping time than AG.

Table II implies two additional cases for which the home-ground theory provides no expectations but useful to consider in any case. In the (high **λ,** low **σ**) case, it is interesting to note that the simulation results are mixed without any significant differences between AG or PB. In the (low **λ**, high **σ**) case, agile generally is better on all measures, but not significantly better. Significance is defined as having more than 25% difference in the average rank for a measure after "convergence" of the simulation (discussed later).

VII. Discovering New Strategies

Given that our simulation has properties and results that are consistent with expectations derived from the home-ground theory, we have a "fair" and justifiable basis for exploring new strategies by comparing them with simulated AG and PB strategies. Recall that the main tenet of the home-ground theory is that for the majority of development efforts, their five critical project characteristics are generally not within either the agile or plan-based home grounds. The advice given is that a "balance" that mixes the two approaches is best - use agile methods for areas whose characteristics are closer to the agile home ground, use plan-based methods for areas that are closer to its home ground.

Taking this advice with respect to our more narrow interests in requirements prioritization, we would like find a mixture of AG and PB that has its home ground in the middle of the two extremes of Dynamism. That is, we seek a strategy that is effective where a given project is more likely to be – with neither high nor low Dynamism. Furthermore, it is difficult (perhaps impossible) to accurately predict at the outset what the Dynamism for a given project will be, so we seek a strategy that does not depend on knowing this. Furthermore, there is no justification assuming that Dynamism is constant throughout the development. Hence the ideal strategy does not depend on knowing what the Dynamism actually is and will "adjust" automatically if it changes.

There are a number of interesting ways to mix PB and AG. The primary trade-off consideration is effort. Adding plan-based activities to an agile approach or vice-versa will increase the effort needed to perform prioritization. This is another consideration when comparing new strategies. If a mixed strategy has better performance, it should be large enough to outweigh the additional effort required use it. With these in mind, two candidate mixed strategies are:

The AG2 strategy: We note that the PB strategy takes advantage of Pareto optimization when prioritized by cost-benefit. This approach is effective in low Dynamism projects by ensuring that regardless of when the development ends, the majority of high-leverage requirements get implemented first i.e. the "80%" of the value that resided in "20%" of the requirements (quotes added to emphasize that these are only representative percentages). Clearly in a high Dynamism environment where values may change radically, what may once have been a high-leverage requirement may transform to a low-leverage requirement (or vice-versa) thereby obviating the benefit of this prioritization. Hence a natural variation of AG is to order the requirements in each iteration by cost-benefit (or value/cost in our simulation) rather than by value alone. The effort increase is that a cost assessment must now be performed in addition to the usual value assessment. We will call this strategy AG2.

The HY strategy: The AG strategy is effective when the Dynamism is high because it enables adjustment of priorities when requirement values change. The shortcoming of this

strategy is that it assumes that always going with the highest valued requirements that are known within each iteration will result in implementing the highest valued requirements overall. That is, local value maximization will lead to global value maximization. Generally the "local optimization leads to global optimization" property does not hold and the result is a sub-optimal prioritization with respect to the overall development (i.e. over all the iterations). This is also true regardless of what local prioritization method is used e.g. cost-benefit rather than value. As such, the AG2 strategy will suffer this shortcoming as well and so we seek an alternative. The basic problem is that the agile method iteration sizes do not account for the likelihood of higher leverage requirements appearing in a later iteration. Indeed, agile methods tend to have fixed iteration sizes (usually fairly small) with the goal to implement as much as possible in that iteration then add the remainder to the next iteration. The small fixed size iterations are intended to keep the development from wandering too far off course when there is high Dynamism.

The PB strategy avoids the agile shortcoming by making an exhaustive identification and assessment of requirements from the outset and then optimizes by prioritizing high leverage requirements first and then tries to control Dynamism as the development progresses. The assumption is that it is very unlikely that a high-leverage requirement will ever become low-leverage. This assumption tends to hold fairly well, however controlling Dynamism is not necessarily achievable in general, which is why the PB strategy is risky in a high Dynamism environment. As an alternative, a mixed strategy can be formed that performs an initial comprehensive requirements assessment to establish an overall initial prioritization (order is by highest to lowest cost-benefit), and then for subsequent iterations is updated according to re-assessments of cost and value and new requirements. The key is in choosing a strategic iteration size that implements the majority of the high-leverage requirements and then leaves the lower-leverage ones for the next iteration where they can be re-prioritized relative to new requirements and updated costs and values. A natural strategic criterion for this is when the total cost exceeds the total value for the requirements implemented (this is the so-called "economic turning point"). The iteration size will vary depending on the particular costs and values of the unimplemented requirements - typically large at first, then progressively decreasing as the cost-benefits of the unimplemented requirements become more equal. This approach assumes that there is a reasonably significant variation in the costs and values of the requirements to reduce the risk of implementing a low-leverage requirement in favor of a high-leverage one simply because of a fixed iteration size. A strategy that performs in this way we will call the "hybrid" or HY. The strategy is straightforward to apply and a brief example can be found in the Appendix.

VIII. SIMULATION RESULTS

We begin by visualizing a single simulation trial under a typical Dynamism scenario and then under two extremes to validate the expected behavior of the simulation for AG and PB and observe the adaptability properties of the new AG2 and HY strategies. Figure 1 is a single trial with λ=1.4 and σ=15% (medium Dynamism).

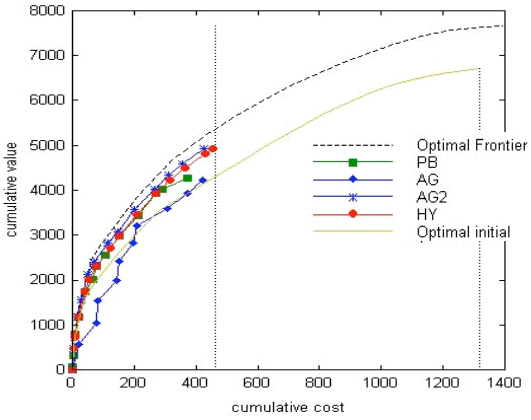

Figure 1. Medium Dynamism simulation trial

Each of the curves in the figure represents the results of applying a given strategy to the final set of requirements (with the exception of the Optimal initial curve). Each point on a curve represents the implementation of a requirement at which time its cost is added to the cumulative total and its value is added to the cumulative value up to that point. Thus strategy curves are non-decreasing. To explain the features in the figure above, the solid curve is the "Optimal initial" strategy and it is the Pareto plot of the base set of requirements. If there is no Dynamism, this would be the optimal strategy as no other ordering could have higher value at lower cost than this curve at any point. The dashed curve is the "Optimal Frontier" which represents the Pareto plot of the set of requirements at the time development ends. This includes the base set plus any requirements added during the development and their respective value changes. No strategy can rise above this curve as it represents the optimal strategy given retrospect knowledge of all the requirements and their values. Such a strategy is impossible to achieve because of the stochastic nature of requirements change. However, the more similar a strategy is to the Optimal Frontier, the better is its performance, and hence the motivation for the **fr** measure. The difference between the Optimal initial and Optimal Frontier indicate the degree of Dynamism for a given trial i.e. the further apart these are, the greater the Dynamism. The tall vertical line indicates the maximum total cost of the development at the stopping time. No strategy may exceed this cost, and so no curve may extend past this line.

We see that all the strategies are consistent with the features described above and so we have further confirmation that the simulation is correct. This verification has been performed hundreds of times under a large variation in simulation parameters with no unexpected behavior. We observe that neither AG nor PB perform particularly well in this "typical" non-extreme Dynamism case as is predicted by the home ground theory. The AG2 and HY perform best and at about the same level.

Figure 2 is a single trial with **λ**= 1/1000 and **σ**=1/10% (low Dynamism). For this scenario we would expect that since there is nearly zero Dynamism, the Optimal initial and Optimal Frontier would be identical. This is clearly the seen in the figure (the two curves completely overlap). With no Dynamism, the PB strategy is predicted to be best and AG the worst. In fact, PB should be identical to the Optimal initial up to the stopping time. This is indicated in the figure above but is difficult to see as it is covered over by the HY strategy. This latter observation is notable as it indicates that the HY strategy has precisely adapted to the expected best strategy PB. This result has been seen to hold in general, and so we have confidence that HY is adaptable to Dynamism. It is also notable that the AG2 strategy performs reasonably well and appears to be adaptable. A more subtle observation is that the AG2 and AG plans both clearly show the "diminishing returns" characteristic within each fixed iteration whereby the values sharply increase at the beginning of each new iteration (the humps). This is consistent with the expected behavior of the agile approach.

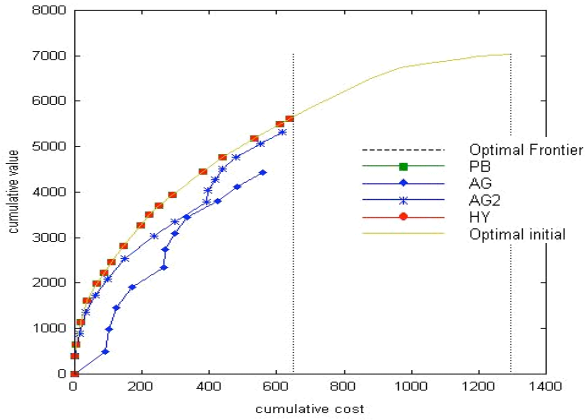

Figure 2. Low Dynamism simulation trial

Figure 3 is a single trial with **λ**= 20 and **σ**=200% (very high Dynamism). With many new requirements and large value swings, we expect a large difference between the Optimal initial and Optimal Frontier as seen in the figure. Also expected from this, and clearly seen in the figure, is that AG would greatly outperform the PB strategy. In this trial, all the strategies that re-prioritize within each iteration seem to perform equally well (at least at the end development time). This is not generally the case, and even here closer inspection will reveal significant differences in these strategies. For example, if the development ended much earlier, say at 500, then the AG2 strategy would be the clear winner. Indeed it is consistently closer to the Optimal Frontier than the other strategies, so its **fr** measure will be higher. What is notable is the similarity of the AG2 and HY strategies. Both strategies appear to adapt well at this extreme level of Dynamism. In this particular case better than AG, but in general this is not true.

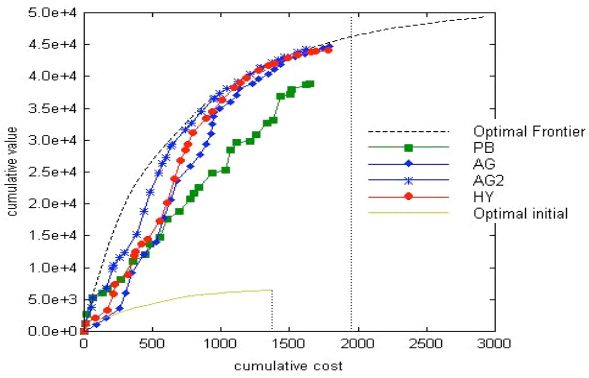

Figure 3. High Dynamism simulation trial

To get a handle on the properties of the strategies generally, we consider the average ranks and standard deviations over 1000 simulation trials for the six strategy measures described earlier. We chose 1000 trials because uniformly, the average values converged to at least 1 decimal of precision for all the measures at this level. The rank for a measure *M* on a given trial is the (*number of strategies being compared*) - (*number of strategies with "worse" measures within the specified tolerance*). For example, if there are 4 strategies being compared and the Agile strategy has a **tv** that is more than 5% (the tolerance level) greater than the other 3 than it will have rank 1. It is unjustifiable to consider two values that are within a given small percentage of each other to have different ranks, hence the use of a tolerance value (we use 5%) to create rank-equivalences. Thus it is possible for all strategies to have the same rank indicating no substantial differences for that measure.

Table III below is an example of average rank results for 1000 trials with **λ**=1.4 and **σ**=15% (medium Dynamism).

TABLE III. AVERAGE RANKS N=1000, Λ=1.4, Σ=15%

	tv	**tc**	**int**	**ben**	**cb**	**fr**
PB	1.67	1.18	1.82	1.74	1.49	1.90
std	*1.04*	*0.46*	*0.97*	*1.08*	*0.87*	*0.86*
AG	2.34	1.26	3.43	2.45	2.41	3.56
std	*1.32*	*0.57*	*0.94*	*1.32*	*1.32*	*0.84*
AG2	1.18	1.21	1.42	1.20	1.16	1.34
std	*0.55*	*0.51*	*0.70*	*0.57*	*0.48*	*0.62*
HY	1.07	1.23	1.23	1.09	1.10	1.21
std	*0.30*	*0.53*	*0.51*	*0.35*	*0.40*	*0.47*

The results are significant to 1 decimal place and we see that the HY strategy is the top rank for all but **tc** (as is expected from our previous discussion on cost and PB). Also notable is that HY has the lowest standard deviation for all measures where it is top ranked. This can be interpreted as the degree to which we expect a typical simulation trial to be near its average rank value. That is, the HY strategy is more consistently top ranked than the other plans. It is not top

ranked on average because it swings wildly from top to bottom rank.

Table IV considers the top average ranks from 1000 random trials under different levels of Dynamism. The PB strategy wins on **tc** mostly due to the abrupt stopping rule which is that no requirements can be implemented that exceed the cost at the stopping point. What happens with the plan-based strategy is that at the stopping point, the planned set of requirements generally cannot exactly meet the cost limit at the stopping time (no rearrangements are possible in the plan-based strategy, even at the end) so there is no choice in which requirement to end on – the one which makes the cumulative cost less than the stopping point whose next requirement in the planned order would make a cumulative cost exceed the stopping point. Since the requirements are ordered by cost-benefit, the farther the stopping time is, the greater the cost per requirement is likely to be. So the end requirement is more likely to "pull back" far behind the stopping point cost because the next requirement is likely to have a higher cost. Because of this, total cost is probably not a very useful "goodness" of strategy measure.

In the overall spectrum of performance, HY dominates in 4 areas, is very strong in another 2, and is strong, but not dominant in 1 additional area from a total of 7 of the 9 scenarios considered. It is worth noting that the strongest areas are when the Dynamism is medium in either **λ** or **σ**, and curiously when **λ** is low and **σ** is high**.** The dominance of HY is indicated in Table IV by the shading where darker shades indicate greater HY dominance.

TABLE IV. AVERAGE RANKS FOR N=1000 TRIALS

	LOW σ=0%	MED σ=15%	HIGH σ=200%
HIGH λ=20	*Value:* AG2 *Cost:* PB *Integral:* AG2 *Ben:* AG2 *CB:* AG2 *FR:* AG2	*Value:* AG2 *Cost:* PB *Integral:* AG2 *Ben:* AG2 *CB:* AG2 *FR:* AG2	*Value:* AG *Cost:* PB *Integral:* HY *Ben:* AG *CB:*HY, AG *FR:* HY
MED λ=1.4	*Value:* HY *Cost:* PB *Integral:* HY *Ben:* HY *CB:* HY *FR:* HY	*Value:* HY *Cost:* PB *Integral:* HY *Ben:* HY *CB:* HY *FR:* HY	*Value:* HY *Cost:* PB *Integral:* HY *Ben:* AG *CB:* HY *FR:* HY
LOW λ=0	*Value:* HY, PB *Cost:* PB *Integral:* HY *Ben:* HY, PB *CB:* PB *FR:* HY, PB	*Value:* HY *Cost:* PB, AG2 *Integral:* HY *Ben:* HY *CB:* HY *FR:* HY	*Value:* HY *Cost:* HY *Integral:* HY *Ben:* HY *CB:* HY *FR:* HY

HY: Hybrid, PB: Plan-based, AG: Agile, AG2: Agile cost-benefit

IX. CONCLUSIONS AND APPLICATIONS

In the essential area of requirements prioritization, we have provided empirical support for the home ground theory of Boehm-Turner [2] that suggests, for most projects, a mixed agile (AG) and plan-based (PB) strategy is best. It is not clear how to construct such a strategy. Using simulation we have discovered two (to the best of our knowledge) new mixed strategies which we call AG2 and HY that, at least within our simulation adapt very well to any Dynamism level. The best performing strategy is HY but at increased effort over PB. AG2 also performs well and requires less effort than PB but more than AG. We have high confidence in our results because the assumptions, properties and behaviors of the simulation are consistent with reliable empirical studies. Although these studies did not provide the detail required to determine all the appropriate simulation parameters, sensitivity analysis shows that our results are remarkably insensitive to variations in the simulation parameters and even to variations in the basic assumptions.

These results suggest that a project using a predominantly agile approach should use the AG2 strategy, while a more plan-based project should use HY. Both strategies are fairly simple and explicitly defined. The simulation is *constructive* in that it actually performs the prioritizations as if it were a non-simulated project. Only the requirements and their evolution are simulated. While we did not detail the implementations of the HY or AG2 strategies, to apply them on an actual project, one simply need follow the steps from what the simulation does. The simulation source code can be obtained at:

http://db-itm.shidler.hawaii.edu/port_research/req_pri/

The additional benefits of reduced variance (*std* in Table III) for the AG2 and HY strategies should not be under appreciated. Reduced variance directly translates into lower risk and more predictable and controllable outcomes. This would be reason enough to utilize hybrid strategies.

In the future we hope to perform some form of follow-up empirical study on the application of HY and AG2 to real projects. Perhaps searching specifically for projects with requirements volatility which are predicted to fit AG2 and HY strategies and study if these projects can actually apply these strategies in practice. Single-case studies, selected on the basis of simulation, would be substantially more valuable than just "any" case study selected by random. The benefit here is that it lends evidence that the particular case study results may hold in general and are not simply anomalous.

Aside from the potentially useful new prioritization strategies, we hope this work will inspire other simulation based investigations of software engineering methods that are intractable to in-vitro experimentation and validation. We believe that simulation can provide meaningful insights into software development micro-processes in ways that theory, empirical studies, and experiments cannot. The assumptions for a simulation should be explicitly formulated and derived from high quality empirical studies. The validity and correctness of the simulation follow by checking the consistency of results with theory and known results. Confidence in new results can be had by comparing these with what would be predicted from theory. Performing sensitivity analysis on both the simulation parameters and assumptions enable understanding of the degree of applicability of simulation results to actual practice where many parameters are unknown and perhaps unknowable.

In conclusion, simulation appears to be a practical and useful means for investigating and validating software development micro-processes such as requirements prioritization. It is useful for both helping to justify a micro-process methodology, help investigate possible improvements (or avoidance of a given method), and better understand the factors involved and their possible consequences. It should be a standard tool within automated software engineering research. In this regard, we have begun to develop **SimSWE** – an opensource software engineering research simulation toolbox for Freemat (and other compatible numerical mathematics systems). The components built for the simulation used for this study will be contributed to this toolkit in addition to some guidelines on building simulations for automated software engineering research.

REFERENCES

[1] Beck, K. *Extreme Programming: Explained,* 7th ed. Boston, MA: Addison-Wesley, 2001.

[2] Boehm, B. Turner, R. "Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods," proceedings 26th International Conference on Software Engineering (ICSE), pp. 718-719 (2004)

[3] Barry Boehm, Value-based software engineering, ACM SIGSOFT Software Engineering Notes, v.28 n.2, March 2003

[4] Cao, L., Ramesh, B., Requirements Engineering Practices: An Empirical Study, IEEE Software, Volume: 25, Issue: 1. page(s): 60-67 (2008)

[5] Karlsson, J. Software requirements prioritizing. proceedings ICRE'96, 1996.

[6] Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Software 14 (1997) 67–74

[7] Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. Information and Software Technology 39 (1998) 939–947

[8] Lehtola, L., Kauppinen, M., and Kujala, S. (2004) 'Requirements Prioritization Challenges in Practice', Proceedings of 5th International Conference on Product Focused Software Process Improvement, Kansai Science City, Japan, pp. 497- 508.

[9] Moisiadis, F. The fundamentals of prioritising requirements., In System Engineering, Test and Evaluation Conference, Sydney, Australia, 2002.

[10] Paetsch, F., Eberlein, A., and Maurer, F. 2003. Requirements Engineering and Agile Software Development. In Proceedings of the Twelfth international Workshop on Enabling Technologies: infrastructure For Collaborative Enterprises (June 09 - 11, 2003). WETICE. IEEE Computer Society, Washington, DC, 308.

[11] Port , Kazman, Nakao, Katahira, "Practicing What is Preached: 80-20 Rules for Strategic IV&V Assessment", Proceedings of IEEE Conference on Exploring Quantifiable Information Technology Yields (EQUITY), 2007, Amsterdam, Netherlands

[12] Siddiqi, J., Shekaran, M.: Requirements engineering: The emerging wisdom. IEEE Software 2 (1996) 15–19

[13] Sjoberg, D., et al, Conducting realistic experiments in software engineering, Proceedings of the International Symposium on Empirical Software Engineering 2002.

[14] Wiegers, K.E.: Software Requirements. Microsoft Press, Redmont, Washington (1999)

[15] [15] The Standish Group, The CHAOS Report, http://www.standishgroup.com/chaos.html, 2006.

APPENDIX: EXAMPLE HY STRATEGY

Below is an example illustrating the performance of the HY method with some Dynamism for the case of few initial requirements. We assume that requirements *R1-R7* are the "base" requirements with *R8-R10* emerging in later.

iteration 0 & 1: The known requirements are ordered from highest to lowest CB and then Iteration 1 is defined by the set of requirements with CB greater than the turning point, which is in this case equals 0.95, implemented in the current order i.e. R2, R3, then R5:

Requirement	R2	R3	R5	R6	R1	R4	R7
Value	6	2	27	9	12	3	1
Cost	2	1	20	13	20	5	2
CB	3	2	1.35	0.69	0.6	0.6	0.5

iteration 2: When the requirements in iteration 1 are implemented, iteration 2 begins by removing these from consideration and revisiting. Value of R_4 increased to 10, new requirements R_8 and R_9 emerge, and the new turning point equals 0.97 giving R_4 and R_8 for this iteration:

Requirement	R4	R8	R9	R6	R1	R7
Value	10	17	11	9	12	1
Cost	5	9	13	13	20	2
CB	2	1.9	0.85	0.69	0.6	0.5

iteration 3: R_4 and R_8 are removed, turning point is 0.69

Requirement	R9	R6	R1	R7
Value	11	9	12	1
Cost	13	13	20	2
CB	0.85	0.69	0.6	0.5

iteration 4: R_{10} emerges and the turning point equals 0.68:

Requirement	R10	R1	R7
Value	12	12	1
Cost	15	20	2
CB	0.8	0.6	0.5

iteration 5: CB's are close in value so this is the last iteration:

Requirement	R1	R7
Value	12	1
Cost	20	2
CB	0.6	0.5

The figure below shows the implemented strategy as compared to the frontier. Clearly, in spite of Dynamism, the HY strategy performs quite closely to the optimal. Also, notice the different iteration sizes.