

Running head: STRENGTHS AND LIMITATIONS OF RESPONSIVE AND AGILE METHODS

Strengths and Limitations of Responsive and Agile Methods such as Extreme Programming on
New Product Development, Value Creation, Innovation, and Market Success

David F. Rico

Introduction

Kent Beck invented a new software development methodology while developing a payroll system for Chrysler in 1996. Chrysler hired Kent Beck as a software testing consultant to help speed up their extremely slow payroll system, which took 100 days to issue a monthly payroll. Kent intuitively diagnosed the systemic management problems at the root of the late, over budget, and faulty payroll system, and devised an off-the-cuff prescription for fixing Chrysler's woes. As lead management consultant, Kent Beck successfully brought Chrysler's payroll project to rapid completion inventing the tenets of Extreme Programming along the way. He wrote a paper in a major international journal describing his success in 1999, published a book in 2000, and launched perhaps the most ubiquitous international software development methodology ever conceived.

U.S. corporations spend more than \$250 billion on information technology projects annually. These expenditures result in over 200,000 projects. In 1994, 32% of all U.S. projects failed resulting in \$81 billion in losses, and 53% had at least a 200% cost overrun. By some estimates, two-thirds or 66% of all projects in the U.S. apply Extreme Programming, which amounts to nearly 132,000 projects. It is not surprising that project failures have decreased by 50% to only 16% of the total number of projects reducing project losses by \$40 billion. While there is no direct correlation between Extreme Programming and the 50% increase in project success rates over the last 10 years, it is pretty clear that project failures are being arrested in unprecedented numbers and frequency. In fact, these improvements could be attributed to any number of best practices or all of them in combination.

Since the debut of Extreme Programming in 1999, over 60 books have been published on the topic and more journal articles have been written by management researchers and practitioners on this topic than any single software development methodology. Extreme Programming has captured the imagination of the top management scientists in the fields of computer science and software engineering ranging from Barry Boehm and Victor Basili.

The literature brings an unprecedented number of unique perspectives to bear upon Extreme Programming. Management researchers and practitioners have performed comparative and historical studies, devised innovative measures, applied real options and value based management, and evaluated productivity and return on investment. They've conducted attitudinal and market penetration surveys, suggested theoretical knowledge management models, and probed for weaknesses and areas for improvement. The most interesting studies demonstrated when it is most appropriate to use it and when it is not, described big bang deployment of Extreme Programming in monolithic organizations, applied formal change management principles, and designed sophisticated evaluation frameworks.

Literature Review

Abrahamsson, Salo, Ronkainen, and Warsta (2002) analyzed 10 software development methods against criteria for agile methods. Their purpose is to lay a much-needed foundation for further academic and industrial research and practice. They compared Extreme Programming, Scrum, Crystal Clear, Feature Driven Development, Rational Unified Process[®], Dynamic Systems Development Method, Adaptive Software Development, Open Source Software, Agile Modeling, and Pragmatic Programming. The primary criteria for agile methods consist of an emphasis on individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

Abrahamsson, Warsta, Siponen, and Ronkainen (2003) identified and analyzed nine major agile methods. Their purpose is to determine the practicality of agile methods and investigate areas for further research and improvement. They compared them on the basis of life cycle coverage, project management support, practical guidance, fitness for use, and empirical evidence. They presented a timeline and history of agile methods and a graphical comparison. They call for more project management support, better descriptions of life cycle context, development of detailed definitions, and a focus on improving the quality versus the quantity of agile methods.

Anderson (2004) identifies and proposes seven metrics for agile methods. His purpose is to distinguish and differentiate agile methods from traditional software development methods. He describes how agile methods improve market competitiveness through value creation, productivity gains, good corporate governance, alignment with shareholder interests, process and value chain transparency, efficient resource allocation, and the best use of shareholder funds. The metrics are design process model, design-in-process inventory, cumulative flow model, drum buffer type, critical chain for probabilistic project scheduling, throughput accounting, and constraint monitoring.

Beck (1999) introduced an agile method for software development called Extreme Programming. His purpose was to create an unconventional life cycle to improve the outcome of software development projects by managing ambiguous customer requirements, estimation, project planning, scheduling, and change. Its 13 practices are planning game, small releases, metaphor, simple design, tests, refactoring, pair programming, continuous integration, collective ownership, on-site customer, 40-hour weeks, open workspace, and just rules.

Erdogmus and Favaro (2003) apply the advanced economic principles of real options to Extreme Programming. Their purpose is to show how Extreme Programming creates more economic value than traditional software development methods. They distill its 13 practices to a single theory of inherent flexibility, which is used to

create value under uncertainty. They describe economic models for discounted cash flows, net present value, call options, Black-Scholes formula, and risk neutral valuation.

Erdogmus and Williams (2003) identify and apply metrics and models for determining the economics of Extreme Programming practices, such as pair programming. Their purpose is to demonstrate that Extreme Programming increases productivity by 543 times over conventional software development approaches and is twice as fast as its nearest competitor. They introduce economic models for breakeven unit value ratio, net present value, deferred realized value, marginal rework cost, total discounted cost, marginal value earned, incrementally realized value, maximum discounted cost, and breakeven unit value in continuous delivery. They also introduce numerous primitive metrics, which establish the basis for their advanced economic models.

Favaro (2003) identifies basic metrics and models for evaluating the economics of Extreme Programming. His purpose is to encourage managers to apply the principles of value based management to Extreme Programming using Stern Stewart's Economic Value Added (EVA[®]) model. He introduces models for net present value and economic profit. He also describes how Extreme Programming focuses on value creation and suggests the application of real options for evaluating its economics.

Johnson (2002) conducted a survey to measure attitudes with agile methods among international programmers. His purpose was to determine and report the experiences of international software developers with agile methods. He designed a 10-question survey to measure experience levels, business impact, and market direction with respect to agile methods. Based on 131 responses 85% of respondents indicated high experience levels, 50% reported significant cost reductions, 93% reported productivity increases, 88% reported quality increases, 83% reported increased customer satisfaction, and 95% plan on continuing use of Extreme Programming.

Kahkonen and Abrahamsson (2003) investigate, analyze, and identify the theoretical basis for Extreme Programming as a principle agile method. Their purpose is to suggest that the effects well-established knowledge management models explain and justify the advantages of using Extreme Programming. They identified two models of knowledge management that explain the benefits of Extreme Programming. Candidate models included the Socialization-Externalization-Combination-Internalization (SECI) and the Articulate-Appropriate-Learn-Act-Accumulate-Anticipate (5-A) models for knowledge management. Their analysis suggests that the 5-A model of knowledge management best explains the benefits of Extreme Programming. They suggest the use of knowledge management models such as 5-A as a theoretical basis for future research concerning Extreme Programming.

Lindstrom and Jeffries (2004) identified and analyzed coarse-grained characteristics of five major software development models. Their purpose was to distinguish, differentiate, and identify the advantages of agile methods and Extreme Programming from traditional software development models. They compared and contrasted Capability Maturity Model Integration[®] (CMMI[®]), structured analysis/structured design (SA/SD), Rational Unified Process[®] (RUP[®]), agile methods, and Extreme Programming. They assert that values, principles, and practices are the most important characteristics of software development models, thus using them as a basis for analytical criteria. Agile methods and Extreme Programming are the only models that have values and principles, according to their analysis. And, they report that Extreme Programming is the only model with explicit practices.

Lippert, Becker-Pechau, Breitling, Koch, Kornstadt, Roock, Schmolitzky, Wolf, and Zullighoven (2003) recommend adding 11 more practices to Extreme Programming and agile methods. Their purpose is to modify and adapt Extreme Programming for long-term and large-scale software development projects with complex domains and limited resources in order to enhance security and reliability without limiting the advantages of agile methods. They divided their 11 additional practices into 6 broad classes: planning for multiple customers, customer testing, complex or imprecise application domain, long-term planning, dependencies among project tasks, and requirements definition. The additional practices are product managers, phased parallel construction, distribution over multiple teams, preparing the customer for the testing effort, more up-front risk management, providing documentation and feedback on test results, author-critic cycles, customer-on-demand, exploratory phase with subsequent rough release schedule, baselines with responsibilities, and business stories.

Maurer and Martel (2002a and 2002b) analyze the productivity increases associated with using Extreme Programming as an agile methodology. Their purpose is to demonstrate that Extreme Programming increases productivity by 213 times over conventional software development methods, while exhibiting only two-thirds the productivity increase of its nearest competitor. Their results were obtained from an analysis of a small software company consisting of nine developers, which produced a web-based application over a 16-month period. They analyzed lines of code, number of Java methods, and number of Java classes per hour as the basis for productivity.

Muller and Padberg (2002 and 2003) created an economic model for evaluating the benefits of using Extreme Programming. Their purpose was to demonstrate that Extreme Programming has economic advantages over conventional software development methods only on larger and more complex software projects. Conversely, they found that Extreme Programming has no economic advantages on small and simple software projects. Their

economic model is a composite of seven metrics and models. The metrics and models consist of net present value, development time, number of pairs, defects left, defect difference, quality assurance time, and development cost.

Spayd (2003) describes a case study of implementing Extreme Programming in a large-scale telecommunications company. He has a two-fold purpose of presenting a model for organizational change involving Extreme Programming and presenting lessons learned from implementing Extreme Programming among 5,000 computer programmers. His model of organization change consists of eight determinants. They are obtain committed executive leadership, articulate the burning platform that demands change and the vision of where you're going, form and maintain a highly effective change team, adhere to key project and change management practices, keep stakeholders engaged throughout the life of the project, align organizational elements and neutralize politics, empower affected employees in adopting the new system, and do not lose focus. His lessons are top-down change does not work well with agile methods, Extreme Programming does not scale to the enterprise level, middle management buy-in is necessary, establish rigid performance benchmarks, and use facilitators or coaches.

Succi, Pedrycz, Marchesi, and Williams (2002) conducted a survey to measure attitudes among international computer programmers towards the Extreme Programming principle of pair programming. Their purpose was to demonstrate that Extreme Programming and pair programming enhance job satisfaction within the international community. They designed a 21-question survey and received 108 responses over a six-month period. Half of the respondents used Extreme Programming and pair programming and half did not. 19% of pair programmers were very satisfied, while only half as many non-pair programmers were very satisfied. 30% of pair programmers were satisfied, while 32% of non-pair programmers were satisfied. Only 9% of the total number of pair programmers and non-pair programmers were unsatisfied or very unsatisfied.

Williams, Krebs, Layman, Anton, and Abrahamsson (2004) designed a generalized framework for evaluating the effectiveness of Extreme Programming among software development organizations. Their purpose is to create an empirical body of knowledge to further the cause of Extreme Programming among researchers and practitioners. Their framework consists of three broad classes of factors: context factors, adherence metrics, and outcome measures. Context factors consist of sociological, project-specific, ergonomic, technology, and geographic factor subclasses. Adherence metrics consist of planning, testing, and coding subclasses. Each of the subclasses is broken down even further. The objective is to determine the context in which Extreme Programming is being applied, the level of compliance, and the business impact of applying its practices.

Research Issues

Research questions. Several research questions have emerged. Does Extreme Programming result in lower costs? Does Extreme Programming result in higher productivity? Does Extreme Programming result in higher quality? Does Extreme Programming result in greater economic benefit? Does Extreme Programming result in higher net present value? Does Extreme Programming result in higher return on investment? Does Extreme Programming result in shorter cycle time? Does Extreme Programming result in greater product volume? Does Extreme Programming result in greater product variety? Does Extreme Programming result in higher revenue? Does Extreme Programming result in higher profit? Does Extreme Programming result in higher shareholder value?

Interest to researchers and practitioners. By some estimates, global information technology budgets will grow by 15 times to nearly four trillion dollars, increasing the number of software projects, and stimulating the demand for Extreme Programming or superior alternatives. Management researchers and practitioners need to understand its theoretical tenets, structural strengths and weaknesses, and performance characteristics. 21st century software development will be automated, obsolescing manually-intensive methods like Extreme Programming.

Significance and impact of research. The disciplines of computer science and software engineering are in their infancy compared to civil engineering, which has been around for millennia. Management scientists are just beginning to apply advanced measurement principles to industrial age models of software development. And, it is clear from the literature survey that the early quantitative studies, in spite of their sophistication, can be easily disproved. Formal deductive studies to test early conclusions will be of enormous significance and impact.

Sustainability during doctoral program. The best studies of Extreme Programming have only emerged in the last 12 to 24 months. A dissertation focusing on Extreme Programming will continue to be viable over the course of the next two years, and yield innovative results. The dissertation can even segue way into more advanced methods of software development and use Extreme Programming as a spring board to future approaches. None the less, its principles of value creation versus time-in-motion based methods may be the key to better methods.

Feasibility of addressing research questions. The first three research questions concerning cost, productivity, and quality can be addressed very easily. These are the tenets of the industrial age software development models, which Extreme Programming is designed to replace. Traditional management scientists understand how to analyze these economics with far greater ease, understanding, sophistication, completeness, and validity than the novice researchers and practitioners, which conducted the early quantitative studies of Extreme Programming. Economic benefits, net present value, and return on investment are dependent variables against the independent variables of cost, productivity, and quality. These are easily determined. Data on cycle time, product volume, and product variety would be a little harder to obtain. However, these data may be obtainable through industry analysis, literature surveys, or new surveys of key firms. Revenue, profit, and shareholder value are publicly available and easily obtainable. The only challenge would be to discover which firms are using Extreme Programming, and then collect their corporate performance data. Corporate trends would have to be analyzed.

Assessment Against Dissertation Criteria

Passion. I definitely have a track record of passion for performing comparative analyses of software development methodologies from both a management and technical point of view. Furthermore, I have a talent for linking software development methodologies to business goals and objectives like Extreme Programming.

Management domain. A study of the performance characteristics of Extreme Programming certainly falls into the domain of engineering management. Classical theorists link management to organizational leadership, while engineers struggle to link engineering management principles to the role of organizational leadership. Chief information officers will certainly benefit from understanding the economic principles of Extreme Programming..

Significance. The significance of an analysis of Extreme Programming can only be understood in the context and size of the global information market, which today exceeds \$300 billion and will increase by an order of magnitude in the next decade. Organizational leaders certainly want to know why failed projects cost their shareholders \$40 billion in losses in the U.S. during 2004, and how Extreme Programming reduces these losses.

Durability. The economics of engineering management, and hence Extreme Programming and Agile methods, are timeless. Economic metrics, models, and measurements are transferable and interchangeable with a variety of information technologies. Minimally, this dissertation will provide an interesting historical retrospective.

Doability. A study of the strengths and weaknesses of Extreme Programming can be initiated almost immediately using historical data, as a process of designing and validating a research methodology and approach. Once the research methodology is in place, comprehensive literature surveys, collaboration with international researchers, simple experiments, or new attitudinal data can be collected to exercise the research methodology.

Applicability. Quantitative and qualitative studies of the strengths and weaknesses of Extreme Programming, agile methods, and other software development approaches are in short supply. There are never enough economic studies of software development approaches. Existing studies are need of replication or refutation.

Originality. A study of Extreme Programming from a traditional point of view is highly original. The literature survey reveals a lack of understanding of traditional views by management researchers and practitioners.

Faculty support. It will be a challenge for the doctoral candidate to link a study of Extreme Programming to the field of engineering management, and then link it to the tenets of classical management theorists.

Timeframe manageability. This is a very simple study by a doctoral candidate with experience conducting comparative analyses of software development approaches and applying economics to engineering management.

Research Questions, Hypotheses, and Key Variables

Table 1

Extreme Programming Outcomes, Measures, Research Questions, Hypotheses, and Key Variables

Outcome	Measure	Type	Research Question and Hypothesis	Dependent Variable	Independent Variable
New Product Development	Cost	Research Question	Does Extreme Programming result in lower costs?	Effort	Training, Process, Design, Test, and Maintenance Effort
		Null Hypothesis	<i>H₀: Extreme Programming does not result in lower costs.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in lower costs.</i>		
	Productivity	Research Question	Does Extreme Programming result in higher productivity?	Effort to Size Ratio	Effort, Software Size
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher productivity.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher productivity.</i>		
	Quality	Research Question	Does Extreme Programming result in higher quality?	Defect to Software Size Ratio	Defects, Software Size
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher quality.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher quality.</i>		
Value Creation	Economic Benefit	Research Question	Does Extreme Programming result in greater economic benefit?	Total Life Cycle Cost	Effort, Quality, Software Size
		Null Hypothesis	<i>H₀: Extreme Programming does not result in greater economic benefit.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in greater economic benefit.</i>		
	Net Present Value	Research Question	Does Extreme Programming result in higher net present value?	Benefit to Discount Rate Ratio	Economic Benefit, Inflation Rate, Time
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher net present value.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher net present value.</i>		
	Return On Investment	Research Question	Does Extreme Programming result in higher return on investment?	Adjusted Benefit to Cost Ratio	Economic Benefit, Costs
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher return on investment.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher return on investment.</i>		
Innovation	Cycle Time	Research Question	Does Extreme Programming result in shorter cycle time?	Calendar Time per Release Cycle	Effort, Elapsed Time
		Null Hypothesis	<i>H₀: Extreme Programming does not result in shorter cycle time.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in shorter cycle time.</i>		
	Product Volume	Research Question	Does Extreme Programming result in greater product volume?	Product Release Frequency	Number of Products, Cycle Time
		Null Hypothesis	<i>H₀: Extreme Programming does not result in greater product volume.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in greater product volume.</i>		
	Product Variety	Research Question	Does Extreme Programming result in greater product variety?	Number of Product Variations	Number of Products, Number of Product Families
		Null Hypothesis	<i>H₀: Extreme Programming does not result in greater product variety.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in greater product variety.</i>		
Market Success	Revenue	Research Question	Does Extreme Programming result in higher revenue?	Sales per Business Unit	Quarterly Earnings, Annual Earnings, Tax Writeoffs
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher revenue.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher revenue.</i>		
	Profit	Research Question	Does Extreme Programming result in higher profit?	Adjusted Sales	Sales, Expenses, Expenditures
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher profit.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher profit.</i>		
	Shareholder Value	Research Question	Does Extreme Programming result in higher shareholder value?	Adjusted Corporate Value	Free Cash Flows, Non Operating Assets, Debts
		Null Hypothesis	<i>H₀: Extreme Programming does not result in higher shareholder value.</i>		
		Alternative Hypothesis	<i>H₁: Extreme Programming results in higher shareholder value.</i>		

References

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis* (478). Oulu, Finland: VTT Publications.
- Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, Oregon, USA*, 244-254.
- Anderson, D. J. (2004). Making the business case for agile management: Simplifying the complex system of software engineering. *Motorola Software, Systems, and Simulations Symposium (S3S 2004), Itasca, Illinois, USA*.
- Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer*, 32(10), 70-77.
- Erdogmus, H., & Favaro, J. (2003). Keep your options open: Extreme programming and the economics of flexibility. In M. Marchesi, G. Succi, J. D. Wells, & L. Williams (Eds.), *Extreme programming perspectives* (pp. 503-552). New York, NY: Addison Wesley.
- Erdogmus, H., & Williams, L. (2003). The economics of software development by pair programmers. *The Engineering Economist*, 48(4), 283-319.
- Favaro, J. (2003). Value based management and agile methods. *Proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), Genova, Italy*, 16-25.
- Johnson, M. (2002). *Agile methodologies: Survey results*. Victoria, Australia: Shine Technologies.
- Kahkonen, T., & Abrahamsson, P. (2003). Digging into the fundamentals of extreme programming: Building the theoretical base for agile methods. *Proceedings of the 29th Euromicro Conference (Euromicro 2003), Belek-Antalya, Turkey*, 273-280.
- Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information Systems Management*, 21(3), 41-52.
- Lippert, M., Becker-Pechau, P., Breitling, H., Koch, J., Kornstadt, A., Roock, S., Schmolitzky, A., Wolf, H., & Zullighoven, H. (2003). Developing complex projects using XP with extensions. *IEEE Computer*, 36(6), 67-73.
- Maurer, F., & Martel, S. (2002a). Extreme programming: Rapid development for web-based applications. *Internet Computing*, 6(1), 86-90.
- Maurer, F., & Martel, S. (2002b). *On the productivity of agile software practices: An industrial case study*. Retrieved September 20, 2004, from University of Calgary Web site: <http://sern.ucalgary.ca/~milos/papers/2002/MaurerMartel2002c.pdf>
- Muller, M. M., & Padberg, F. (2002). Extreme programming from an engineering economics viewpoint. *International Workshop on Economics-Driven Software Engineering Research (EDSER), Orlando, Florida, USA*.
- Muller, M. M., & Padberg, F. (2003). On the economic evaluation of XP projects. *Proceedings of the Ninth European Software Engineering Conference (ESEC 2003), Helsinki, Finland*, 168-177.
- Sliwa, C. (2002). Users warm up to agile programming. *Computer World*, 36(12), 8-8.
- Spayd, M. K. (2003). Evolving agile in the enterprise: Implementing XP on a grand scale. *The First Annual Agile Development Conference, Salt Lake City, Utah, USA*, 60-70.
- Succi, G., Pedrycz, W., Marchesi, M., & Williams, L. (2002). Preliminary analysis of the effects of pair programming on job satisfaction. *Proceedings of the Third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002), Alghero, Sardinia, Italy*, 212-215.
- Williams, L., Krebs, L., Layman, L., Anton, A. I., & Abrahamsson, P. (2004). Toward a framework for evaluating extreme programming. *Eighth Conference on Evaluation and Assessment in Software Engineering (EASE 2004), Edinburgh, Scotland, UK*, 11-20.