IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4231

The Case For A Less Methodical Methodology: Lean, Light, Extreme, Adaptive, Agile and Appropriate Software Development

John Mendonca
Purdue University, Indiana, Tel: (765) 496-6015, Fax: (765) 496-1212, jamendonca@tech.purdue.edu

ABSTRACT

Historically, the approach to software engineering has been based on a search for an optimal methodology—that is, the identification and application of a set of processes, methods and tools that can predictably lead to software development success. Less methodical methodologies, under a variety of names, takes a contingency oriented approach. Because of the limitations in the nature of methodology, the high failure rate in software development, the need to develop methodology within an environmental context, and the pressures of fast-paced "E" development, further exploration and definition of a more flexible, contingency-based approach to methodology is justified.

INTRODUCTION

Despite the high rate of failure in software development, the fundamental strategy for achieving quality in software engineering continues to be methodology—that is, discovery and application of that ideal set of processes and practices that lead to software products that are accurate, effective and are delivered on time and within budget. The path to an optimal methodology leads theorists and practitioners toward increasingly refined sets of concepts, models, rules, project management strategies, descriptions of deliverables, tools, testing standards, test case constructs, and the many other components of a well-defined methodology. Perhaps because of its close identity with the "engineering" paradigm, ubiquitous failure seems not to have shaken faith in the methodical approach to software development. In fact, the response to failure seems often to be more methodology.

In recent years, due to the increasing complexity of the information technology (IT) arena and the furious pace of E-commerce and E-business development, a less methodical approach to software development management has gained attention. This approach has often been linked with Extreme Programming (XP) and has been called by a variety of names, including "lean" and "light" methodology (Yourdon, 2000 [1]). Highsmith (2000) used the term "adaptive" in his book describing the basic concepts, but he and others prominent in XP theory and practice seem to have settled on "agile" as the preferred term. Ealier this year, with the support of XP proponents and others, the "Manifesto for Agile Software Development" (2001) was developed and published.

Regardless of the name, the approach embodies two characteristics. The first characteristic is that it is less methodical. It is not fixated on the search for an optimal methodology, but rather is contingency oriented, allowing for adaptation and flexibility depending on environmental issues. The second characteristic is that it incorporates a concept of appropriateness. A methodology must not only adapt to its environment, it must also reflect an appropriate level of regidity, the "just right" level between no methodology and a heavily restrictive one that suffocates rather than informs.

This paper argues that because of the inherent limits to methodology, unrealized expectations, and the fast-paced, complex and unpredictable environment, a less methodical, contingency approach to software engineering is justified.

METHODOLOGY: EXPECTATIONS AND LIMITATIONS

As noted above, a software development methodology is a set of processes and techniques for the management of software development. The numerous formal documented methodologies and many more informal ones vary based on the many paradigms and variables that are part of the software development landscape. Ivaria (2000) suggests there are over 1000 information systems development methodologies and offers a schema for their characterization and evaluation. All methodologies have the common characteristics of being a defined set of activities and are based on the concepts of quality and engineering in software development. In addition, there are two very significant aspects of the nature of methodology itself: first, it is defined, developed and verified only through experience, *after* development has occurred; and second, methodology is itself a *system*.

Pressman (2000) suggests that a methodology is composed of three parts: processes, methods, and tools. Processes provide the framework for activities. At the highest level they prescribe guiding principles, the use of resources, a definition and hierarchy of sub-processes, the sequential order of activities, and other constraints. Methods provide the implementation techniques within the framework of processes. Examples of methods include requirements analysis procedures, design paradigms, testing strategies and program construction procedures. Tools, such as computer aided software engineering (CASE) and project management software, support processes and methods.

Methodology is at the core of the concept of "software engineering" (SE). IEEE (1993) defines SE as the application of a "systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software." The engineering paradigm is thus predicated on well defined processes within a generally predictable environment with well defined outcomes and roles, sequential project (phased) development, and historical information for a "best practices" approach. SE is tightly coupled to quality software development via methodology, the means by which quality is achieved. Anyone who has worked with a methodology, however, is accutely aware that it is no *guarantee* of quality. It does provide a framework for proceeding, for organizing and understanding the tasks ahead, so it is a good base from which development can proceed.

However, every methodology is limited in two significant ways. First, a useful methodology is developed only after systems have been implemented, both successfully and unsuccessfully, and successful processes and methods ("best practices") are identified. It took a large body of experimentation, knowledge and practice, for example, before the parameters of the Systems Development Life Cycle (SDLC) were well defined and stabilized. The object paradigm, embodied in object-oriented (OO) analysis, design, and programming, had a profound impact on systems development methodology (Capper, 1994). OO methodologies continue to undergo rigorous experimentation, testing, and proof (see the CETUS links website [18,452 Links] for information and links to over 50 documented OO methodologies). This "lagging" characteristic of methodology is especially significant because developers working in the latest paradigms, such as Internet and middleware

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

development, cannot rely on well-defined methodologies for structure and guidance.

The second significant way in which methodologies are limited is that a methodology is itself a system. A "one size fits all" approach to software development cannot work. Certain processes are appropriate for some conditions and inappropriate for others. Processes differ because environmental factors differ—factors such as the experience of people involved, goals, project scope, time to market, technologies used, and many others. This concept is significant because it recognizes the responsibility of developers for selecting an appropriate methodology, or even appropriate sub-components of many different methodologies, in order to support and promote development success. It also highlights the flexible and adaptive nature of appropriate methodology, especially in dynamic, emerging environments where change is a dominant characteristic.

METHODOLOGY APPLIED

The case for a less methodical methodology is based on three assertions: 1) the failure of methodology to provide consistent success in SE; 2) the lack of well-developed, well-defined methodologies applicable to a fast paced, innovative, emergent systems development environment; and 3) the need for using an appropriate approach to SE that balances the demands of effectiveness with efficiency—a level of methodology that is "just right."

On Failure and Methodology

In the introduction to his widely used text on software engineering, software quality guru Roger Pressman (1997) refers to what he calls a three-decades long "chronic affliction", that affliction being the ongoing problems associated with software development and the continuing high rate of failure. Despite investments in software engineering, management frameworks and development methodologies, failures still litter the IT landscape. The reasons for failure are numerous (see Lientz [2000] for an excellent annotated list) and the application of methodology *per se* certainly should not be universally offered as the *only* critical factor in success or failure. In fact, defining "failure" is itself not a simple task. For example, is a six-month project that comes in two weeks late a failure; or one that returns its investment in 36 months instead of the expected 32?

Popular estimates that claim a fifty percent or greater failure rate are not easily verified, but anecdotal evidence is abundant. The Standish Group's CHAOS survey (Johnson, 1999), for example, claims that only 26% of 1998 projects were deemed "successful" by survey respondents. Whatever the specific rate, most executives and information technologists would probably agree that failure is significantly more common than one should expect considering the great investment in software "engineering" concepts and discipline over the past decades. However failure is defined, or whatever failure rate is accepted, the point is that methodology, as a quality assurance construct that is expected to ameliorate development problems, if not eliminate them, has regularly failed to do so.

Methodology's role in failure may be the result of several kinds of errors. One explanation for failure might be an inappropriate selection of methodology among the variations available—that is, a mismatch between the methodology and the characteristics of the development environment. In addition to this selection failure, another possibility is the misapplication of an appropriate methodology. A third possibility, one that is increasingly common in an emergent technical environment, is the situation in which there is no appropriate methodology on which developers can rely. This "lagging" characteristic of methodology may contribute to failure, or at least does not assist in the avoidance of failure, because in an innovative technical environment proven methodologies are not defined and documented. The beginning points of large software development paradigm shifts are especially susceptible to this kind of failure.

Software development for enterprise integration is arguably a current example of this problem. In the past five years the pressure to integrate existing processes and systems (rather than building new ones) or to implement packaged software (rather than developing in-house versions) has grown tremendously. A survey by Morgan Stanley states that 35 percent of Fortune 500 companies list integration as their top objective (Sulllivan, 2001). Other listed objectives, such as E-business and Customer Relationship Management, also have a critical integration component and, when included, would tend increase that percentage significantly. The Boston Consulting Group (Dickel, 2000), in a survey of more than 100 CEO's and CIO's involved in enterprise-wide systems implementations, reported that only 33 percent of integration projects were viewed as "positive" in terms of value creation, cost effectiveness and financial impact. Based on investment and expectations, this is an arena in which successful methodologies are much needed, but are sorely lacking.

Methodology in a Fast-Paced, Emergent Environment

IT is arguably the most disruptive force to organizations in the past century, being both a driver and an object of organizational change. In its role as chief enabler of better-faster-cheaper for the organization, IT is a critical success factor for delivering competitive advantage within the rapidly changing business environment. E-commerce and E-business place a particular strain on methodology, requiring "trade-offs between schedule, functionality, resources and quality" (Yourdon, 2000 [2]) in an extremely demanding environment. A contemporary software development methodology, therefore, will be required to operate in an environment with the following characteristics (adapted from Mendonca, 2000):

- A rapid pace in introduction of new technologies (software and hardware).
- A demand for expeditious development and implementation, leading to new rapid development and implementation techniques.
- Telecommunications integrated into, and inseparable from, the computing environment.
- Modularization of hardware and software, emphasizing object assembly and processing.
- Integration of seemingly incompatible diverse technologies.

In this kind of emergent, unformed, somewhat chaotic environment, a traditional approach to using an optimal methodology—predictable, tested, and proven—has no application.

There is no doubt that even in environments where the pace is not so rapid and change not so prevalent, methodology is often suspected among business proponents, users, and some IT non-managerial staff to be overly rigorous, bureaucratic, and burdensome to the point of being a hindrance rather than an enabler. It is viewed as being even more so in a fast-paced, emergent environment.

Appropriate Methodology: Just Enough

There are numerous ways to judge the success or failure of any one particular software development effort. Measurements commonly identified include effectiveness (does the product function as expected?); value (does the product meet its financial and other value-enhancing goals?); cost (is the cost justified by the benefits accrued?, is the project within budget?); and timeliness (was the project delivered on time?). As noted above, these parameters play against each other—that is, as we increase resources to ensure factors such as effectiveness and timeliness, we increase costs associated with the project, and therefore erode efficiency valuations. A recognition of this tension in the business environment is important to IT and non-IT staff alike.

Having no methodology at all risks chaos and possible delivery of a product that doesn't work or is over budget or has taken too long to implement. On the other hand, a highly structured, rigid methodology, what James Highsmith very descriptively calls a "monumental" methodology (Highsmith, 2000) may be too costly because of the use of unnecessary resources. Organizations faced with time-sensitive E-business projects that demand quick implementation for competitive advantage must carefully consider the options. The challenge is, of course, to identify how much is "just right." Under some circum-

stances, a less rigid methodology may be more appropriate, while under other circumstances, for example in a well-defined environment with known deliverables, a more rigid methodology is a good choice.

The SDLC model, as embodied in formal commercially available methodologies developed by consulting companies, is certainly an example of the optimized, monumental approach to methodology. Another example, arguably, is the Software Engineering Institute's Capability Maturity Model (CMM) for Software, which has become widely recognized within the IT industry. It includes a detailed description of processes in the context of software development process improvement (Paulk, 1995). Although it is presented primarily as a framework for improvement, rather than a methodology *per se*, CMM defines multiple "key process areas" (methods) that organizations adopt to develop full capability in quality software development. However, its formal procedures, detailed documentation and heavy resource requirements have come under for inapplicability to small and medium-sized projects and development in fluid environments. The result has been that some proponents are now advocating using "CMM with good judgment" to soften its inflexibility (Paulk, 1999).

A systems approach to methodology considers environmental factors that determine not only the appropriate selection of processes, methods, and tools, but also the appropriate level of formality to be applied. These factors include:

1) Technology factors: What technologies are being used? Are they new or mature? Have successful appropriate methodologies been developed that provide a good fit for the technology?
2) People factors: What methodologies are developers experienced with? Is there a good expectation of strong developer/user collaboration?
3) Project definition factors: Is the project well defined in scope and objectives or will it require definition as it unfolds? What is the expected development timeline? Can the project be subdivided in mini-projects with smaller deliverables?
4) Processes: Are business processes stable or does the project require process re-engineering?

Yourdon (2000) calls this approach a "risk/reward" approach to defining an appropriate level of resource investment in methodology. A chaotic non-methodology development environment oriented to "code and fix" is not an acceptable alternative; neither is an inflexible monumental methodology in many circumstances. Fowler (2000) suggests that a flexible, less methodical approach to methodology attempts a "useful compromise between no process and too much process, providing just enough to gain a reasonable payoff."

TOWARD A LESS-METHODICAL METHODOLOGY

What does a less methodical methodology look like and how does it respond to the requirements described above? Indeed, is a light/lean/extreme/adaptive/agile/appropriate methodology a methodology at all? A contingency approach to methodology recognizes the limitations of methodology, the experience of failure, and the inherent difficulty in creating optimized methodologies in a rapidly changing emergent environment. Its approach is to choose the processes and techniques appropriate to a given environment, to take advantage of the capabilities of any and all formalized methodologies, and to apply guidelines for developing processes that may not have been fully defined.

The definition of this approach to software development continues to evolve, with many participants from the project management, quality assurance, and software engineering disciplines joining together to contribute. Developers of the XP concepts, notably Kent Beck and Martin Fowler, have played an important role by way of adopting similar principles for flexible, adaptive programming techniques. Other development frameworks, such as Crystal methods, Scrum, Adaptive Software Development, and Dynamic Systems Development Methodology have also contributed (Highsmith, 2001). James Highsmith's book on adaptive software development, recent writing on "agile" development, and the Manifesto form the core of these principles and guidelines.

Fowler (2000) very accurately describes these related methodologies as fundamentally "adaptive rather than predictive" and "people-oriented rather than process-oriented." While it is not the objective here to fully define the nature of these methodologies, basic principles include:

- Agile processes that continuously respond to changes in the environment.
- Appropriate selection of process components that reflect efficiency in addition to effectiveness.
- An adaptive approach (frameworks) rather than adherence to predefined process rules.
- Frequent, rapid delivery of smaller software components to achieve faster feedback.
- A collaborative approach to development.
- An expectation of change during the development process.
- Outcomes are emergent, rather than fixed.
- Creativity in problem solving.
- Dynamic re-prioritization.

The principles and guidelines of a flexible, less methodical methodology continue to be described and defined. It is clear that this is a genuinely different approach than the optimized one inherent in a fixed and fully defined methodology. Practices such as continuous feedback and incremental product delivery will form the core of techniques for implementation of this approach. However, there clearly needs to be further work in identifying those frameworks and practices that support the concept and can support successful delivery of quality software. Somewhat ironically, like any other methodology, it will need to be tested and proven. Considering the limitations of methodology and the increasing complex environment in which software is engineered, however, it is good potential and is worthy of our attention.

REFERENCES

18,452 Links on Objects and Components. Website: http://www.cetus-links.org**.**

Capper, N.P. and R.J. Colgate, 1994. "The Impact of Object-Oriented Technology on Software Quality." IBM Systems Journal, January.

Dickel, K. and Sirkin, H. 2000. "Getting Value from Enterprise Initiatives: A Survey of Executives". Boston Consulting Group, Inc. Available at: http://www.bcg.com/publications/files/Enterprise_computing_report.pdf.

Fowler, M. 2000. "Put Your Process on a Diet." Software Development, December.

Highsmith, J.A., 2000. Adaptive Software Development: a Collaborative Approach to Managing Complex Systems. Dorset House.

Highsmith, J.A., Cockburn, A., and Boehm, B., 2001. "Agile Software Development: The Business of Innovation." Computer, September.

IEEE, 1993, "Standards Collection: Software Engineering", IEEE Standard 610.12-1990.

Iivari, J., Hirschheim, R., and Klein, H.K., 2000/2001. "A Dynamic Framework for Classifying Information Systems Development Methodologies and Approaches." Journal of Management Information Systems, Winter.

Johnson, J., 1999. "Turning Chaos into Success." Software Magazine, December.

Lientz, B.P., and Rea, K.P. On Time Technology Implementation. London: Academic Press.

"Manifesto for Agile Software Development." available: http://www.agilealliance.org**.**

Mendonca, J., 2000. "Educating the Business Information Technologist: Developing a Strategic IT Perspective." Proceedings of the Information Resources Management Association.

Paulk, M., *et al*, 1995. The Capability Maturity Model: Guidelines for Improving the Software Process. Pittsburg, PA: Carnegie Mellon University, Software Engineering Institute.

Paulk, M.C., 1999. "Using the CMM With Good Judgment." Software Quality Professional, June.

Pressman, R. S., 1997. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Companies, Inc.

Sullivan, T., 2001. "Take Your Medicine." Inforworld, August 10.

Yourdon, E., 2000. "The 'Light' Touch." Computerworld, September 18.

Yourdon, E., 2000, "Success in E-Projects." Computerworld, August 21, 2000.