A REVIEW ON LIGHTWEIGHT METHODOLOGIES

Zuriani Ahmad Zukarnain, Roslan Jamaludin and Nor Zalina Ismail
Faculty of Information Technology and Quantitative Sciences, Universiti Teknologi MARA Pahang

ABSTRACT

System Development Life Cycle methodologies are traditional mechanisms to assure that software systems meet established requirements. Traditional methodologies sometimes fall short in the new e-business environment. They are often to "heavy" to keep up with the pace of e-business software development projects. In response to this problem, "light" SDLC methodologies have recently been developed and widely used. This paper begins with a brief overview of traditional SDLC processes. This is followed by an in-depth look at new lightweight methodologies. In this discussion, methods such as Adaptive Software Development (ASD), Agile Software Process (ASP), Crystal, Dynamic System Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Rational Unified Process (RUP), SCRUM, and Whitewater Interactive System Development with Object Models (Wisdom) were explored. The paper concludes with a summary of lightweight SDLC methods along with recommendations for their use.

INTRODUCTION

Traditional SDLC methodologies sometimes fall short in the new e-business software environment (Yourdon 2000). They are often to "heavy" to keep up with the pace of e-business software development projects. In response to this problem, "light" SDLC methodologies have recently been developed and widely use. They are considered light because of the reduced documentation and managerial effort required.

System Development Life Cycle (SDLC) methodologies are mechanisms to assure that software systems meet established requirements (DOJ 2000). These methodologies impose various degrees of discipline to the software development process with the goal of making the process more efficient and predictable.

This paper was a research-based descriptive study. The primary research method of writing this paper was browser-based Internet searches. The literature reviewed included textbooks, journal articles, and magazine articles referenced by a select set of online resources. The full text articles from journals and magazines were located and subsequently downloaded.

The objective of this paper is to identify the strengths and weaknesses of these new lightweight methodologies and to make recommendations for their effective use. For the purpose of discussion in this paper, SDLC methodologies are divided into two groups, traditional and lightweight.

SDLC METHODOLOGIES : AN OVERVIEW

Traditional SDLC Methodologies

Rothi and Yen (1989) provided a brief review of traditional SDLCs. In their journal article, they related how the use of traditional software development models is numerous and often regarded as the proper and disciplined approach to the analysis and design of software applications. Examples of such models included the code and fix, waterfall, staged and phased development, transformational, spiral, and iterative models.

A related article by the U.S. Department of Justice described how the primary goal of any SDLC is to deliver quality software systems (DOJ 2000). It further defined a quality system as one that: 1) meets or exceeds customer expectations; 2) works well with current and planned infrastructure; and 3) is inexpensive to maintain and enhance. SDLC is traditionally made up of several phases, each comprised of multiple steps. These steps typically include software concept, requirements analysis, architectural design, coding and debug, and system testing.

Osborn (1995) also discussed traditional SDLC techniques and how over time the phases of these approaches have become enshrined in a development cycle that includes: 1) defining requirements, 2) designing a system to meet those requirements, 3) coding, and 4) testing. Each phase of the development cycle is strictly sequenced. For example, in the waterfall model, the output of all prior effort is a prerequisite for subsequent steps.

Lightweight SDLC Methodologies

In response to traditional approaches to software development, new lightweight methodologies have appeared (Fowler 2000). A high percentage of software development efforts have no process and might best be described as a chaotic "code and fix" activity. Light SDLC techniques are a compromise between no process and too much process. In the following sections, literature related to nine types of lightweight SDLC methodologies is discussed. These are Adaptive Software Development (ASD), Agile Software Process (ASP), Crystal, Dynamic System Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Rational Unified Process (RUP), SCRUM, and Whitewater Interactive System Development with Object Models (Wisdom).

ASD

Highsmith (1999) focused on the adaptive nature of new SDLC methodologies. His work with traditional predictive methodologies led to the conclusion that they were inherently flawed, in particular when applied to modern business processes. In response, he offered a new approach, Adaptive Software Development (ASD) as a framework from which to address the rapid pace of many software projects. ASD is grounded in the science of complex adaptive systems theory and has three interwoven components: the Adaptive Conceptual Model, the Adaptive Development Model, and the Adaptive (leadership-collaboration) Management Model.

In contrast to the typical waterfall (plan, build, implement) or the iterative (plan, build, revise) life cycles, the adaptive development life cycle (speculate, collaborate, learn) acknowledges the existence of uncertainty and change and does not attempt to manage software development using precise prediction and rigid control strategies (Highsmith 1999). There are six basic characteristics of an adaptive lifecycle (Highsmith 2000). The process is mission focused, component based, iterative, timeboxed, risk driven, and change tolerant.

Adaptive lifecycles are mission focused (Highsmith 2000). Although the final results may be fuzzy in the initial phase, the overall mission is well defined. In addition, adaptive lifecycles are component based in the context that a group of features are developed, i.e. results, not tasks, are the focus. The process is also iterative because it emphasizes "re-doing" as much as "doing." Another characteristic of the practice is timeboxing, i.e. setting fixed delivery times for projects. Timeboxing forces ASD project teams and their customers to continuously re-evaluate the project's mission, scope, schedule, resources, and defects.

Lastly, adaptive lifecycles are risk driven and change tolerant (Highsmith 2000). Similar to the spiral development model, adaptive cycles are guided by the analysis of critical risks. In addition, ASD is tolerant to change. The ability to incorporate change is viewed as a competitive advantage (not as a problem).

The benefits of ASD include the following (Highsmith 1999):

- Applications are a closer match to customer requirements due to constant evolution;
- Changing business needs are easily accommodated;
- The development process adapts to specified quality parameters;
- Customers realize benefits earlier;
- Risk is reduced; and
- Confidence is established in the project early on.

Finally, the ASD methodology's most important benefit is that it forces developers to more realistically estimate their ability. Lack of complete knowledge is assumed and comprehensive feedback mechanisms are built to compensate.

ASP

The Agile Software Process (ASP) was first proposed at the 1998 International Conference on Software Engineering in Kyoto Japan (Aoyama 1998a). Unlike traditional software process models based on volume, the ASP is time-based and quickly delivers software products. The model accomplishes this by integrating lightweight processes, modular process structures, and incremental and iterative process delivery. The ASP methodology offers five major contributions to the field. These include:

- A new process model with a time-based inaction mechanism;
- A software process model that provides evolutional delivery;
- A software process architecture that integrates concurrent and asynchronous processes;
- A process-centered software engineering environment; and
- Experience and lessons learned from the use of ASP for the five-year development of a large-scale software system.

However, ASP is a complex process and is, therefore, more vulnerable to disruption than are other lightweight and traditional SDLC methodologies (Aoyama 1998b).

The benefit of the ASP process is its ability to efficiently manage large-scale software development efforts (Aoyama 1998b). Evidence of this is the 75 percent reduction in development cycle time realized by Fujitsu when ASP was employed to manage a major communication software project.

Crystal

The Crystal family of lightweight SDLC methodologies is the creation of Alistair Cockburn (Fowler 2000). Crystal is comprised of more than one methodology because of Cockburn's belief that differing project types require differing methodologies. Project types are classified along two lines: the number of people on the development team and the amount of risk (e.g. a 30 person project that is at risk to lose discretionary money requires a different methodology than a four person life-critical project).

Crystal methodologies are divided into color-coded bands (Cockburn 2001). "Clear" Crystal is the smallest and lightest. "Yellow", "Orange", "Red", "Maroon", "Blue", and "Violet" follow for use with larger groups using more complex methodologies. Each color has its own rules and basic elements. Each methodology is as light as possible and is tuned to the current project using techniques developed by Cockburn. These techniques are based on the following four principles:

- Use larger methodologies for larger teams;
- Use denser methodologies for more critical projects;
- Interactive, face-to-face communication is most effective;
- Weight is costly;
  For example, Crystal "Clear" is the lightest band and is suitable for use with development teams of up to six people. Rules for this band are (Cockburn 2001)
- Required roles are sponsor, senior designer, designer, and user;
- There is only one team;
- Software is delivered through quality assurance every three months;
- Fewer work products are required;
- Work product templates, coding style, and other standards are determined by the team; and
- Individual techniques are determined by the individual and the team.

Finally, Crystal methodologies are based on the premise that people issues can easily determine a project's results (Cockburn 2000). Software development methodologies should recognize this and take the characteristics of people into account. People are not just nameless resources as is the case with many traditional SDLC methods.

DSDM

The Dynamic Systems Development Method (DSDM) is a framework used to control software development projects with short timelines (DSDM, 2001). It was developed in 1994 (and is currently maintained) by a consortium formed by a group companies in Great Britain. The methodology begins with a feasibility study and business study to determine if DSDM is appropriate. The rest of the process consists of three interwoven cycles. These are functional model iteration, design and build iteration, and implementation.

The underlying principles of DSDM include frequent deliveries, active user communication, empowered development teams, and testing in all phases of a project (DSDM 2001). DSDM is different than traditional approaches in that requirements are not fixed. Project requirements are allowed to change based upon a fixed timeline (i.e. time boxed) and fixed project resources. This approach requires a clear prioritization of functional requirements. Emphasis is also put on high quality and adapting to changing requirements.

Finally, although DSDM is primarily used only in Great Britain, it has the advantage of a solid infrastructure (similar to traditional methodologies), while following the principles of lightweight SDLC methods (DSDM 2001).

XP

Extreme Programming (XP) is the best known of the lightweight methodologies (Highsmith 2000). Proponents are clear about its appropriateness to varying types of development efforts. For example, XP works best when applied to small, co-located teams (less than ten people). XP's basic approach is similar to most iterative rapid application development (RAD) methods. This includes three-week cycles, frequent updates, dividing business and technical priorities, and assigning stories.

XP has four key values: communication, feedback, simplicity, and courage (Highsmith 2000). These build into a dozen practices that are followed during XP projects. The twelve XP practices are: planning, small releases, metaphor, simple design, refactoring, testing, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, and coding standards. One notable difference between XP and other methodologies is its emphasis on testing. Testing is the foundation of all development. In fact, XP programmers are required to write tests as they write production code. Since testing is integrated into the build process, a highly stable and expandable product is the result.

Finally, XP is designed to allow small development teams to deliver quickly, change quickly, and change often (Highsmith 2000). XP provides the absolute minimum set of practices that enable small, co-located development teams to function effectively in today's environment of rapid development.

FDD

Feature Driven Development (FDD) is a model-driven short-iteration software development process (Coad 1999). The FDD process starts by establishing an overall model shape. This is followed by a series of two-week "design by feature, build by feature" iterations. FDD consists of five processes: develop an overall model, build a features list, plan by feature, design by feature, and build by feature.

There are two types of developers on FDD projects: chief programmers and class owners (Coad 1999). The chief programmers are the most experienced developers and act as coordinator, lead designer, and mentor. The class owners do the coding. One benefit of the simplicity of the FDD process is the easy introduction of new staff. FDD shortens learning curves and reduces the time it takes to become efficient.

Finally, the FDD methodology produces frequent and tangible results. The method uses small blocks of user-valued functionality. In addition, FDD includes planning strategies and provides precision progress tracking.

RUP

The Rational Unified Process (RUP) works well with cross-functional projects (Bloomberg 1999). Published by Rational Software, RUP contains six best practices: manage requirements, control software changes, develop software iteratively, use component-based architectures, visually model, and verify quality. RUP is a process framework and can be used in either a traditional (e.g. waterfall style) or a lightweight manner. One example of the model's flexibility is the dX process developed by Robert Martin. The dX process is identical to XP and is a fully compliant instance of RUP. The process was designed for developers that have to use RUP, but would prefer to use XP.

Finally, although RUP was originally intended to help manage software projects, its flexible design makes it applicable to large e-business transformation projects (Bloomberg 2001). After applying a few critical augmentations to the process, RUP can effectively provide a framework for enterprise-wide e-business transformation.

SCRUM

A scrum is a Rugby team of eight individuals (Rising & Janoff, 2000). The team acts together as a pack to move the ball down the field. Teams work as tight, integrated units with a single goal in mind. In a similar manner, the SCRUM software development process facilitates a team focus. SCRUM is a light SDLC methodology for small teams to incrementally build software in complex environments. SCRUM is most appropriate for projects where requirements cannot be easily defined up front and chaotic conditions are anticipated.

SCRUM divides a project into sprints (iterations) of 30 days (Rising & Janoff 2000). Functionality is defined before a sprint begins. The goal of the process is to stabilize requirements during a sprint. Management continues during a sprint with short (fifteen minute) meetings each day. These meetings, called scrums, are where the team decides what it will do the next day.

Finally, SCRUM is an incremental, timeboxed development methodology with a unique feature: daily meetings where three questions are asked (Rising & Janoff 2000). What was completed since the last meeting? What got in the way? What specific things will be completed before the next meeting?

Wisdom

The Whitewater Interactive System Development with Object Models (Wisdom) addresses the needs of small development teams who are required to build and maintain the highest quality interactive systems (Nunes & Cunha 2000). The Wisdom methodology has three key components:

- A software process based on user-centered, evolutionary, and rapid-prototyping model;
- A set of conceptual modeling notations that support the modeling of functional and nonfunctional components; and
- A project management philosophy based on tool usage standards and open documentation.

Wisdom is comprised of three major workflows: requirements workflow, analysis workflow, and design workflow. In addition, the methodology is based on seven models and uses four types of diagrams.

Task flow plays an important role in Wisdom and corresponds to a technology-free and implementation-independent portrayal of user intent and system responsibilities (Nunes & Cunha 2000). This guarantees that the user interface accurately reflects the actual structure of use.

Finally, Wisdom is intended for use by small development teams and has effectively demonstrated its capabilities in several small software companies in Portugal (Nunes & Cunha 2000). In those companies, Wisdom's user-centered focus has resulted in improved usability, efficiency, user satisfaction, and reduced cost and complexity.

SDLC METHODOLOGIES : AN EVALUATION

Strengths and Weaknesses

A recent study by the Cutter Consortium found that traditional SDLC methodologies "fall short in the new e-business environment. They are unable to keep up with fast-paced, ever-changing e-business projects" (Cutter 2000). Perhaps the greatest strength of the new lightweight methodologies is that they provide an alternative to the code and fix mentality that permeates today's environment (Fowler 2000). Simpler lightweight processes are more likely to be followed than traditional ones when a developer is used to no process at all. In addition, lightweight methods excel when requirements are uncertain and volatile. Traditional processes require stable requirements in order to have a stable design and follow a planned process. Another advantage of lightweight methodologies is that they force developers to think clearly about the end products they are developing.

On the other hand, one of the biggest limitations of lightweight SDLCs is their inability to handle large development teams (Fowler 2000). Although XP and Crystal have been successful with teams of 45 to 50, there is no evidence on how to use a lightweight method beyond that number.

Case study on the usage of lightweight methodology

1. Chrysler Compensation System (C3)

On the positive side is the example of the Chrysler Compensation System(C3) (Highsmith 2000). After a 26-person development team failed to complete what was considered a large system that required heavy SDLC, an eight-person team using XP successfully completed the project in one year (Cockburn 2000). The project began in the mid-1990s and developed in Smalltalk. In 1996, the project was in trouble due to a low quality code base. In response, the code was discarded and the project was restarted from scratch using XP as its methodology. Following this redirection, the first phase of C3 went live in early 1997. Currently, the object oriented (OO) payroll system consists of 2,000 classes and 30,000 methods.

2. Motor Insurance System

Another example is Motor Insurance System which is developed by our team (Zuriani Ahmad Zukarnain, Roslan Jamaluddin and Nor Zalina Ismail.). This project was funded by the Institute of Research Development and Commercialization, Universiti Teknologi MARA. The project began in the early 2003. We developed the system by using traditional methodology. In the middle of the year, the project was stuck due to the repetitive user requirement. Every time the requirement changed, we had to restart the project. After one year we decided to use Dynamic System Development Methodology (DSDM) since this methodology is flexible. The project was restarted in February 2004 and completed in August 2004. At the end of the year, we produced a system that meets the user requirement without delay. By using this DSDM methodology, the system was completed in a short time.

CONCLUSION

Reducing the time-to-market is a way of life for most companies (Sims 1997). Shrinking cycle times are commonplace in the software industry. Software developers strive to reduce cycle times because of the competitive advantages provided and the greater number of products they are able to release in the same amount of time. However, time-to-market concerns have forced many developers to make conscious trade-offs between cost, quality, and features.

The "one size fits all" approach to applying SDLC methodologies is no longer appropriate (Lindvall & Rus 2000). Each SDLC methodology is only effective under specific conditions. Traditional SDLC methodologies are often regarded as the proper and disciplined approach to the analysis and design of software applications (Rothi & Yen 1989). Examples of this include the code and fix, waterfall, staged and phased development, transformational, spiral, and iterative models.

Lightweight methodologies, on the other hand, are a compromise between no process and too much process. These new methods were developed to efficiently manage software projects subjected to short timelines and excessive uncertainty and change.
The strengths of these new light methodologies include their simpler processes and easier acceptance by developers who are only familiar with code and fix techniques. In addition, these lightweight SDLCs aid developers in thinking clearly about the end products they are creating. The disadvantages include their inability to handle large development teams. Lightweight methodologies are most appropriate when there are uncertain and volatile requirements, responsible and motivated developers, and customers who wish to become involved. On the other hand, lightweight methods are inappropriate with teams of more than fifty and/or the project has a fixed scope.

Lightweight methodologies re-examine the traditional assumptions that have historically been made about the commitment of resources to requirements analysis and process improvement (Yourdon 2000). Traditional SDLCs operate on the fundamental assumption that it is worth investing resources to identify a flaw in a process because the process will be used over and over again. On the other hand, lightweight SDLCs recognize that when everything is changing and there is no assurance that processes will be reused that it makes little sense to expend the effort.

REFERENCES

Aoyama, M. (1998a, April 19 - 25). *Agile software process and its experience.* Paper presented at the International Conference on Software Engineering, Kyoto Japan.

Aoyama, M. (1998b). Web-based agile software development. *IEEE Software, 15*(6), 56-65.

Bloomberg, J. (1999, October). Software Methodologies on Internet Time. *Developer.com.* Retrieved March 11, 2001, from the World Wide Web: http://softwaredev.earthweb.com/java/sdjjavaee/article/0,,12396_616711,00.html.

Bloomberg, J. (2001, January). Using the RUP for Enterprise e-business Transformation. *WaveBend Solutions.* Retrieved April 7, 2001, from the World Wide Web: http://www.therationaledge.com/content/jan_01/f_rupent_jb.html.

Coad, P. (1999). Feature-Driven Development. *Object International.* Retrieved April 8, 2001, from the World Wide Web: http://www.togethersoft.com/jmcu/chapter6.PDF.

Cockburn, A. (2000, July/August). Selecting a project's methodology. *IEEE Software, 17*(4), 64-71.

Cockburn, A. (2000, September). Balancing Lightness with Sufficiency. *American Programmer.* Retrieved March 11, 2001, from the World Wide Web: http://www.crystalmethodologies.org/articles/blws/balancinglightnesswithsufficiency.html.

Cockburn, A. (2001). Just-In-Time Methodology Construction. *Humans and Technology.* Retrieved March 11, 2001, from the World Wide Web: http://www.crystalmethodologies.org/articles/jmc/justintimemethodologyconstruction.html.

Copernic (2000). Copernic 2001. Retrieved April 1, 2001, from the World Wide Web: http://www.copernic.com.

Cutter. (2000, October). Light Methodologies Best for E-business Projects. *Cutter Consortium.* Retrieved March 11, 2001, from the World Wide Web: http://cutter.com/consortium/research/2000/crb001003.html.

DOJ (2000, March). The Department of Justice Systems Development Life Cycle Guidance Document. *United States Department of Justice.* Retrieved April 1, 2001, from the World Wide Web: http://www.usdoj.gov/jmd/irm/lifecycle/table.htm.

DSDM. (2001). Method Overview: What is DSDM. *DSDM Consortium.* Retrieved April 8, 2001, from the World Wide Web: http://www.dsdm.org/products/overview_02.asp#What.

Fowler, M. (2000, December). Put Your Process on a Diet. *Software Development Online*. Retrieved March 11, 2001, from the World Wide Web: http://www.sdmagazine.com/articles/2000/0012/0012a/0012a.htm.

Fowler, M. (2000, November). The New Methodology. *ThoughtWorks*. Retrieved March 11, 2001, from the World Wide Web: http://www.martinfowler.com/articles/newMethodology.html.

Highsmith, J. (1999). *Adaptive Software Development*. New York: Dorset House Publishing.

Highsmith, J. (2000, February). Extreme Programming. *Cutter*. Retrieved April 8, 2000, from the World Wide Web: http://www.cutter.com/ead/ead0002.html.

Highsmith, J. (2000, July/August). Retiring Lifecycle Dinosaurs. *Software Testing & Quality Engineering*. Retrieved March 11, 2001, from the World Wide Web: http://www.adaptivesd.com/Articles/Retiring%20LC%20Dinosaurs.pdf.

LexiBot (2001). Our Technology - Results: The LexiBot Expression. *BrightPlanet*. Retrieved April 1, 2001, from the World Wide Web: http://www.brightplanet.com/technology/results2.asp.

Lindvall, M., & Rus, I. (2000, July/August). Process diversity in software development. *IEEE Software, 17*(4), 14-18.

Nunes, N., & Cunha, J. (2000). Wisdom: A software engineering method for small software development companies. *IEEE Software, September/October 2000*, 113-119.

Osborn, C. (1995). SDLC, JAD, and RAD: Finding the Right Hammer. *Center for Information Management Studies*. Retrieved April 7, 2001, from the World Wide Web: http://faculty.babson.edu/osborn/cims/rad.htm#SDLCvsJADvsRAD.

Rising, L., & Janoff, N. (2000, July/August). The SCRUM software development process for small teams. *IEEE Software, 17*(4), 26-32.

Rothi, J., & Yen, D. (1989). System Analysis and Design in End User Developed Applications. *Journal of Information Systems Education*. Retrieved April 7, 2001, from the World Wide Web: http://www.gise.org/JISE/Vol1-5/SYSTEMAN.htm.

Sims, D. (1997). Vendors struggle with costs, benefits of shrinking cycle times. *IEEE Computer, 30*(9), 12-14.

Strigel, W. (2000, February). Is Web Development a New Creature? *Software Productivity Center*. Retrieved March 11, 2001, from the World Wide Web: http://www.spc.ca/resources/essentials/feb2300.htm#three.

Yourdon, E. (2000, October). The Emergence of "Light" Development Methodologies. *Software Productivity Center*. Retrieved March 11, 2001, from the World Wide Web: http://www.spc.ca/resources/essentials/oct1800.htm#3.