ISSN: 1545-679X

Information Systems Education Journal

Volume 4, Number 71 http://isedj.org/4/71/ September 7, 2006

In this issue:

A Study of Software Methodology Analysis:"Great Taste or Less Filling"

Jeffrey L. Brewer
Purdue University
West Lafayette, IN 47907-1421 USA

Kevin C. Dittman
Purdue University
West Lafayette, IN 47907-1421 USA

Gaurav Ghatge
Purdue University
West Lafayette, IN 47907-1421 USA

Abstract: Software project management methodologies that have developed in the past couple of decades have done so to address the endemic problem of software project failures caused, in a large part, by lack of planning and poor execution. Methodologies like Waterfall, Sashimi, Spiral and Agile have all become key tools in a project manager's tool box. With so many methodologies littering the software development domain, it begs the question as to which development methodology is the right one for a particular project. How does a project manager know which methodology available today is the right one to produce satisfactory results? In this paper we address these questions and also how to aid students in their understanding of these choices.

Keywords: project management, methodology, scrum, rational unified process, rup

---

Recommended Citation: Brewer, Dittman, and Ghatge (2006). A Study of Software Methodology Analysis:"Great Taste or Less Filling" *Information Systems Education Journal, 4* (71). http://isedj.org/4/71/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2005:* §2153. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/4/71/**

The **Information Systems Education Journal** (ISEDJ) is a peer-reviewed academic journal published by the Education Special Interest Group (EDSIG) of the Association of Information Technology Professionals (AITP, Chicago, Illinois). • ISSN: 1545-679X. • First issue: 8 Sep 2003. • Title: Information Systems Education Journal. Variants: IS Education Journal; ISEDJ. • Physical format: online. • Publishing frequency: irregular; as each article is approved, it is published immediately and constitutes a complete separate issue of the current volume. • Single issue price: free. • Subscription address: subscribe@isedj.org. • Subscription price: free. • Electronic access: http://isedj.org/ • Contact person: Don Colton (editor@isedj.org)

EDSIG activities include the publication of ISEDJ, the organization and execution of the annual ISECON conference held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

A Study of Software Methodology Analysis: "Great Taste or Less Filling"

Jeffrey L. Brewer
jbrewer@purdue.edu

Kevin Dittman
kcdittman@purdue.edu

Gaurav Ghatge
gghatge@purdue.edu

Department of Computer and Information Technology
Purdue University, West Lafayette, Indiana 47907-1421 USA

Abstract

Software project management methodologies that have developed in the past couple of decades have done so to address the endemic problem of software project failures caused, in a large part, by lack of planning and poor execution. Methodologies like Waterfall, Sashimi, Spiral and Agile have all become key tools in a project manager's tool box. With so many methodologies littering the software development domain, it begs the question as to which development methodology is the right one for a particular project. How does a project manager know which methodology available today is the right one to produce satisfactory results? In this paper we address these questions and also how to aid students in their understanding of these choices.

Keywords: project management, methodology, scrum, rational unified process

1. INTRODUCTION

System development methodologies like Waterfall, Sashimi, Spiral, and Agile have all become key tools for project managers and software developers to aid them in delivering projects on time, within budget, and meeting customer requirements. Unfortunately, many have not used these methodologies affectively nor have they chosen the right methodology to fit the project. The Standish Group, a West Yarmouth, Mass. consulting company, which published their findings in a report entitled "Chaos" for the first time in 1994 and which annually submits new findings is a leader in assessing risk, return on investment, and cost for Information Technology (IT) investments. At present count they have conducted analyses of nearly 30,000 case studies. (http://www.standishgroup.com). In January 2004, the Standish Group released the latest statistics: 15 percent failure rate and 51 percent of projects meet the challenged criteria as stated previously (SoftwareMag.com, 2004). In fact they are not the only group to put out such compelling figures. According to a recent article written by Scott Berinato for CIO.com, nearly three quarters of all IT projects in the Internet era that were conceived in the last seven years have suffered from one or more of the following: total failure, cost overruns, time overruns, or a rollout with fewer features or functions than promised (Berinato, 2001). A study conducted by the Forrester research group states that nearly one-third of all IT projects commenced would be an average of three months late (Hoffman, 2003). The use of standard methodologies has somewhat helped in making sure that the projects undertaken have met with some degree of suc-

cess. According to pmi.org it has become the norm nowadays for software project management to follow some sort of methodology. Examples can be found in the vast repository of the Project Management Institute at http://www.pmi.org/prod/groups/public/documents/info/pir_pmnetour.pdf.

Until recently, the methodologies that dominated the field were methodologies that were derived from well known engineering fields. These methods approached system development in a requirements/design/build paradigm with standard, well-defined processes. Today they are called various names like heavy methodologies or plan-driven methodologies. However, newer methodologies have started making an appearance in software projects. These methodologies unlike the more classical ones are considered to be more agile and more able to adapt to change. They do not focus on a long development cycle but rather on short iterations, lightweight processes and rely heavily on tacit knowledge of the users (Boehm & Turner, 2003). These types of methodologies have come to be known as light or agile methodologies. With so many methodologies littering the software development domain, it begs the question as to which development methodology is the right one for a particular project. How does a project manager know which methodology available today is the right one to produce satisfactory results? Which one do we teach our undergraduate students to use?

"America spends over $275 billion each year on about 200,000 software development projects, many of which fail" (Crawford, 2001). According to the Standish group, the number of project failures had decreased from about 40 percent in 1996 to about 23 percent in 2001 (Berinato, S., 2001). This has been due to a sustained effort by project managers to use standardized project management methodologies to ensure project success. With the increase in use of heavy and light methodologies to ensure project completion success, project managers are facing questions as to which methodology is the right one that best fits their unique project. The purpose of this study is to formulate a decision tree based on a set number of project characteristics that will help guide project managers in making a selection between two of today's more popular methodologies; one being a heavy methodology while the other being a light methodology. This same criteria is used in our undergraduate courses to help students learn the differences and which methodology to use.

There are an abundance of methodologies available for project managers to chose from, which creates a need to help zero in on the right methodology for their projects. Many project management methodologies are promising to help project managers deliver a project in time and under budget. But we frequently hear stories of millions of dollars spent on projects wasted because the wrong methodology was used (Brichter, 1999; Boehm, 2003). These examples indicate that a tool that helps project managers determine the best methodology is of paramount importance. This study analyzed major project characteristics like scope, people, size etc. and indicates which type of methodology (light versus heavy) was suitable based on each characteristic. This study also analyzed one popular light methodology like Scrum and one heavy methodology like the Rational Unified Process and determined based on the characteristics of the project which type of specific light or heavy methodology is the best fit for the particular project in question.

2. PROJECT CHARACTERISTICS ANALYSIS

IT projects exhibit multiple characteristics. For the duration of this study the authors looked at nine critical success factors or characteristics. Each characteristic is dealt with differently depending on the methodology that is being used. These nine characteristics were determined from research material available on project characteristics and personal interviews. The interviews occurred with 3 current project managers from two different large pharmaceutical companies, a consultant currently working as a project manager for a large aerospace company and a consultant working as a project manager for several large healthcare providers. All of the interviewees asked that their names not be used. The top nine characteristics are: size of the team, primary project goals, rate of change present in the environment, planning and control, project communication, handling of requirements, design and development of systems, customer relations, and the organization's culture.

Size of team: According to Kent Beck in his book *Extreme Programming Explained*, "Size clearly matters. You probably couldn't run an XP project with a hundred programmers. Not fifty. Nor twenty. Ten is definitely doable" (Beck, 1999). The agile process seems to work best with small applications. Industry wide consensus is that the tight coordination and shared knowledge generally prevents agile methods with teams over forty. (Constantine, 2001). There are definite exceptions to the norm. But they are few and far between. The highly successful 50-person Singapore lending application, and another successful 250-person, banking application have proved the above consensus to be wrong. However after interviewing the managers of both projects they were in no doubt that using an agile methodology was highly risky in ensuring the success of the project. Other examples have been to develop a corporate portfolio of related applications involving around 800 developers at IDX, a medical information services company (Highsmith, 2002). The methodology used on that particular project was SCRUM. Such a large project had to adapt to traditional plans and specifications in order to deal with the increasingly complex, multidimensional interactions among the project's elements (Boehm & Turner, 2003). However, there have been countless failures in using agile methodologies where size has been over forty.

Conversely, heavy methodologies do a much better job at addressing projects which involve large number of people and which are high in complexity. Since heavy methodologies are highly process and document oriented, they provide for better communication and coordination across large groups. The down-side for such a rigid, document driven methodology is that it requires more time to get the project off the ground. Hence such a methodology will not be very efficient on small projects. Boehm and Turner cite that their Software Steering Committee had recently participated in a 150-person, week long review of the completeness and consistency of thousands of pages of specifications for the U.S. Army/DARPA Future Combat System program (Turner & Boehm, 2003). The specifications dealt with 34 highly complex systems and the specifications were produced by multiple integrated product teams. The authors, in their book *Balancing Agility and Discipline*, categorically stated that there was absolutely no way to handle the problem with agile methodologies and tacit knowledge propagation.

According to Alistair Cockburn, "A larger group needs a larger (heavy) methodology" (Cockburn, 2000). A methodologies primary purpose is to coordinate people and communication flow between them. He states that because a large methodology deals with more roles, work products, reviews and so on a project that will exhibit multiple roles and multiple work products will have to be dealt with using a heavy methodology.

Primary Project Goals: Depending on what the primary project goals are the use of methodologies will differ accordingly. The primary goals of plan-driven methodologies are predictability, stability and high assurance. A heavy methodology like SW_CMM or RUP focuses on process improvement. It does so by increasing process capability through standardization, measurement and control. Prediction is based on the measurements of prior standard activities while control is asserted when current progress is outside the expected tolerances (Boehm & Turner, 2003).

For projects which are safety critical and demand high assurance a heavy methodology seems to be a better fit. Such projects require a documented set of plans and specifications and need to adhere to certain government standards. Government standards like RTCA DO-178B require strict adherence to process and specific types of documentation to achieve safety or security. For example, failure in an atomic power plant is more serious than failure in bowling-match tracking software. Accordingly, the methodology the developers use in building the power plant software needs to be more document-driven stable and predictable. Suppose both projects incorporate a requirements gathering technique called "use cases". The bowling league might write them in a few sentences on the board, on a scrap of paper, or in a word processing document. The power plant team will insist on writing them using a particular tool and filling in particular fields. They will call for version control, reviews, and sign-offs at several stages in the life cycle. The benefit is that more writers and readers will be able to collaborate and fewer mistakes will be made,

which is supposed to justify the extra cost. Hence a more critical system- one whose undetected defects will produce more damages – needs more publicly visible correction in its construction (Cockburn, 2000). Proper documentation, extensive communication and more collaboration between the concerned people needs to happen. This can be done with a heavy methodology rather than a light methodology.

If the primary goals of the project are to address change rapidly and be proactive to change then light methodologies seem to be better able to adapt than bureaucratic and rigid plan driven methodologies. Light methodologies do not rely overtly on plan-driven goals and heavy documentation, rather they build things quickly and find out through experience what activity or feature will add the most value next (Boehm& Turner, 2003).

Rate of change present in the environment: Heavy methodologies work best when the requirements are largely determinable in advance and remain stable (Boehm & Turner, 2003). Heavy methodologies can handle concerns like enterprise, organizational, product line etc. across multiple projects. They do so because they cover a broad spectrum of activities and to better handling such a board spectrum of activities they predict future needs through architectures and extensible designs (Boehm & Turner, 2003). Heavy methodologies develop capabilities in related disciplines and impact a large number of people at various levels within the organizational hierarchy.

Light Methodologies "are most applicable to turbulent, high-change environments," and have a world view that organizations are complex adaptive systems, in which requirements are emergent rather than prespecifiable (Cockburn & Highsmith, 2001). Light methodologies generally focus their scope on the matter at hand rather than on problems that might be encountered in the future. They are not necessarily concerned with how the project integrates with an organization's overall infrastructure or scale in production (Silwa, 2002). In general light methodologies are better used when projects are built in-house or in dedicated development environments (cebase.org, 2002).

Planning and Control: Heavy methodologies are very plan-oriented. Hence in some areas they are also called plan driven methodologies. Planning forms an integral part of such methodologies. These types of methodologies rely heavily on documented process plans (schedules, milestones, procedures) and product plans (requirements, architectures, and standards) to keep everything coordinated (Boehm & Turner, 2003). Heavy methodologies also draw heavily from historical data and past performance in planning for current projects. In fact, in such methodologies progress is always tracked against plans. Projects that are highly complex and can span multiple activities will require exhaustive planning of all activities to make coordination between people more dependable.

In Light methodologies, planning is seen as a means to an end rather than a means of recording text. Most of the planning is done in the form of deliberate group planning effort rather than being a part of the methodology. In other words planning is a very ad hoc part of the methodology and a very informal one. If some unforeseen occurrences take place team members call upon each others tacit understanding of the project goals and try and implement a reworked solution.

Project Communication: One of the tenants of heavy methodologies is that they rely primarily on explicit documented knowledge. Communication in heavy methodologies is generally unidirectional i.e. from one entity to another rather than two entities (Boehm & Turner, 2003). Communication procedures are very comprehensive and ensure that all foreseeable situations are communicated.

Light methodologies rely more on person-to-person communication. They are not so concerned with documenting communication between entities. Light methodologies rather rely on tacit, interpersonal knowledge and such methodologies rely on communicating through person-to-person interaction.

Handling of requirements: Plan-driven methods generally prefer formal, base lined, complete, consistent, traceable and testable specifications. Heavy methodologies generally first identify requirements, define requirements and then hand off the software requirements to the appropriate team. Plan

driven methods also focus more on dealing with quality of nonfunctional requirements such as reliability, throughput, real-time deadline satisfaction, or scalability (Boehm & Turner, 2003). Hence a project exhibiting a high level of criticality is bound to follow a plan driven methodology.

Light methodologies express requirements in terms of changeable, informal stories. There is a close interaction between customers and developers to determine the highest-priority set of requirements to be included in each iteration. Light methodologies count on their rapid iteration cycles to determine the needed changes in the desired capability and to fix them in the next iteration (Li & Alshayeb, 2001). Customers express their strongest needs and the developers assess what combinations of capabilities are feasible for inclusion in the next developmental iteration. Negotiations establish the contents of the next iteration (Boehm, 2000). Light methodologies also handle prioritized and evolutionary type requirements better than plan driven methodologies (Beck, 1999).

Design and development of systems: Plan driven methodologies use planning and architecture- based design to accommodate foreseeable change. This effort allows the designers to organize the system to take advantage of software reuse across product lines (Boehm & Turner, 2003). By using plan driven methodologies companies like Hewlett-Packard were able to reduce its software development cycle time from 48 months to 12 months over 5 years, by developing plug and play reusable software modules (Lim, 1998). Hence for a project which has to have a robust architecture, plan driven methodologies are a good fit as predictability and dependability are primary objectives of such methodologies.

Light methodologies advocate simplicity at every turn. The simpler the design the better is one of extreme programming's goals (Beck, 1999). Such methodologies look at only catering to the minimum requirements as set up by the users or customers. They do not anticipate new features and in fact expend effort to remove them. Light methodologies assume that the cost of rework to change software to support new, possibly unanticipated, capabilities will remain low over time (Boehm & Turner, 2003). They also assume that the application situation will change so rapidly that any code added to support future capabilities will never be used (Beck, 1999).

Customer Relations: Customer relations are an area that is addressed differently by different methodologies. Heavy methodologies generally depend on some form of contracts between the developers and customers as the basis for customer relations. They try to cope with foreseeable problems by working through them in advance and formalizing the solutions in a documented agreement. However, having contracts can cause start-up delays since contracts can be a drawn out process. Contracts need to be extremely precise to encompass all the expectations. An incomplete or imprecise contract can lead to unsatisfactory results due to incomplete expectations. The net result is that it may lead to project failure leading to loss of trust and an adversarial relation between the system developer and the customer. Heavy methodologies bank on their process maturity to provide confidence in their work (Boehm & Turner, 2003).

Light methodologies strongly depend on dedicated, collocated, customer representatives to keep the projects focused on adding rapid value to the organization (Boehm & Turner, 2003). However, care has to be taken to ensure that these customer representatives must act as liaisons and there is total synch between the system users they represent and the development team. If this is not the case and if the representatives do not completely understand the user needs or are unable to reflect them to the systems development team there can be total project failure. Light methodologies use working software and customer participation to instill trust in their track record, the systems they have developed and the expertise of their people.

Organizational Culture: Organizational Culture is an important criterion which can relate directly to the success of the methodology that is being used. If the organizational culture dictates that clear policies and procedures will define a person's role then heavy methodologies tend to best fit that organization. Every person's task is well-defined and documented. The general rule is that each developer, programmer, analyst will accomplish the tasks given to him or her to exact specifications so that their work

products will easily integrate into others work products with limited knowledge of what others are actually doing (Boehm & Turner, 2003).

When organizations are not so rigid and where people have more freedom available to them to define their work roles and policies are made as required then light methodologies are a better fit. Each person is expected and trusted to do whatever work is necessary to the success of the project. There is no definite role or scope of work that has to be completed by each person. It is expected that unnoticed tasks are completed by the person who first notices them and so forth.

3. PROJECT METHODOLOGY ANALYSIS

The two representative methodologies selected for this study were Scrum and the Rational unified Process. Each is explained in this section.

Scrum was developed by Ken Schwaber and Jeff Sutherland. Scrum is based on the concept that software development is not a defined process, but an empirical process with complex input/output transformations that may or may not be repeated under differing circumstances (Boehm & Turner, 2003). One of the biggest influences on Scrum is the evolving scientific discipline known as complexity theory. It is concerned with the behavior over time of certain kinds of complex systems. The first reference to the term Scrum in literature can be attributed to the article of Takeuchi and Nonaka as early as 1986. In their article Takeuchi and Nonaka talk about an adaptive, quick, self-organizing, product development process (Schwaber & Beedle, 2002). The name is essentially derived from the game of rugby. In rugby a play where two opposing teams attempt to move against each other in large, brute-force groups is called a Scrum. Each group must be quick to counter the other's thrust and adjust and exploit any perceived weakness without the luxury of planning (Boehm & Turner, 2003).

The Scrum approach has been developed for managing the systems development process. It is an empirical approach applying the ideas of industrial control process theory to systems development resulting in an approach that reintroduces the idea of flexibility, adaptability and productivity (Schwaber & Beedle, 2002). It does not specify or define any software technique in the development phase. Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment. Table 1 lists some of the advantages and disadvantages of using Scrum.

RUP is a Software Engineering Process (Kruchten, 1999). RUP is the direct successor to the Rational Objectory Process. The Rational Objectory Process was the result of the integration of the "Rational Approach" and the Objectory process (version 3), after the merger of Rational Software Corporation and Objectory AB in 1995 (Gornik, 2001). From its Objectory ancestry, the process has inherited its process structure and the central concept of a use case. From its Rational background, it gained the current formulation of iterative development and architecture (Gornik, 2001). Finally, RUP is a specific and detailed instance of a more generic process described by Ivar Jacobson, Grady Booch, and James Rumbaugh in the textbook, The Unified Software Development Process.

RUP provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget. The Rational Unified Process enhances team productivity, by providing every team member with easy access to a knowledge base with guidelines, templates and tool mentors for all critical development activities (Jacobson et al, 1992). By having all team members accessing the same knowledge base, no matter if you work with requirements, design, test, project management, or configuration management, it ensures that all team members share a common language, process and view of how to develop software (Brown, 1996). RUP activities create and maintain models. Rather than focusing on the production of a large amount of paper documents, RUP emphasizes the development and maintenance of models—semantically rich representations of the software system under development (Booch, 1995). Table 2 lists some of the advantages and disadvantages to using RUP. The advantages were taken from several resources

Table 1. Why use Scrum

Advantages	Disadvantages
Productivity increases ○ Some Scrum teams have recorded a 4x increase in productivity (Schwaber, 1997). ○ Most improve productivity by 10-20% depending on management commitment. (Schwaber & Beedle, 2002).	Requires hands-on management, but not micromanagement (Boehm & Turner, 2003). ○ Management must be willing to make changes to help Scrum teams succeed ○ Scrum requires constant monitoring both quantitatively and qualitatively
Continuous improvement ○ Scrum enables continuous, rapid, bottom-up reengineering (Schwaber & Beedle, 2002).	Requires management to delegate decision-making authority to the Scrum team (Beedle et al, 2000).
Leverages the chaos (Schwaber, 1997) ○ The product becomes a series of manageable chunks ○ Progress is made, even when requirements are not stable ○ Everything is visible to everyone ○ Team communication improves ○ The team shares successes along the way and at the end ○ Customers see on-time delivery of increments ○ Customers obtain frequent feedback on how the product actually works ○ A relationship with the customer develops, trust builds, and knowledge grows	Scrum is new and different ○ People are resistant to change ○ Some workers are not comfortable with the responsibility Scrum enables

Table 2. Why use RUP

Advantages	Disadvantages
Risks are mitigated earlier	Not easy to tailor to smaller projects (Kruchten, 2001)
Change is more manageable	
Higher level of reuse	Leads to a tendency to "take it all", which leads to high implementation costs (Kruchten, 1999).
Project team can learn along the way	
Better overall quality	
Enhances team productivity, by providing every team member with easy access to a knowledge base with guidelines, templates and tool mentors for all critical development activities	Has a large volume of process guidelines and is very detail heavy (Boehm & Turner, 2003).

(Boehm, 1996; Booch, 1995; Jocobson, 1999; Pollices, 2003).

4. DECISION TREE ANALYSIS

The decision tree diagrams that follow are separate and distinct decision trees. They do not constitute parts of one large decision tree. Although that might be desirable, it was determined that it was outside the scope of this study. To come up with an overall decision tree consisting of nine project characteristics would be an enormous undertaking. The ranking of the nine characteristics would have to be done by each project manager. It is very difficult to say which characteristic is more important than the other. It boils down to each project and what is being achieved by that project and the concerned project manager. It can only be arrived at by conducting a survey of multiple projects and determining individual weightings for each characteristic.

4.1 Size of Team Characteristic

Figure 1 indicates that if size of the team is more than ten then it is advisable to use a heavy methodology like RUP. The primary reason is that the larger the team is, the more difficult it becomes to coordinate the team and its members. Similarly light methodologies are better at handling smaller teams. Industry wide consensus is that the tight coordination and shared knowledge generally prevents agile methods with teams over forty (Constantine, 2001).

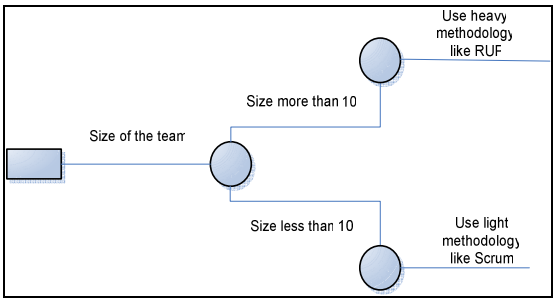

Figure 1: Team Size Characteristic Decision tree

4.2 Primary Project Goal Characteristic

Figure 2 indicates that if the primary goal of a project is to deliver a predictable and stable system then it is better to use a heavy methodology like RUP. Heavy methodologies focus on process improvement. For projects which are safety critical and demand high assurance a heavy methodology seems to be a better fit. If the primary goals of the project are to address change rapidly and be proactive to change then light methodologies seem to be better able to adapt than bureaucratic and rigid plan driven methodologies. Light methodologies do not rely overtly on plan-driven goals and heavy documentation, rather they build things quickly and find out through experience what activity or feature will add the most value next (Boehm & Turner, 2003).

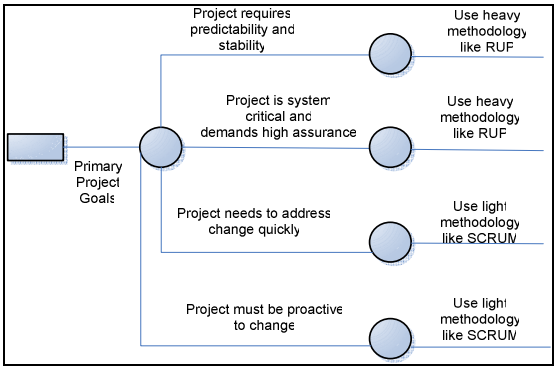

Figure 2: Primary Project Goal Characteristic Decision Tree

4.3 Rate of Change Characteristic

Figure 3 indicates that if requirements are going to remain stable over time and requirements can be determined in advance it is better to use a Heavy Methodology like RUP. Light methodologies are better suited to an environment that is always changing or can be described as turbulent. Light methodologies focus their scope on the matter at hand rather than on problems that might be encountered in the future.

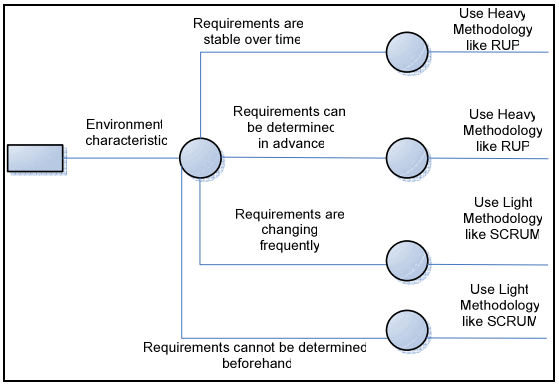

Figure 3: Rate of Change Characteristic Decision Tree

4.4 Planning and Control Characteristic

Figure 4 illustrates that heavy methodologies like RUP rely heavily on documented process plans (schedules, milestones, procedures) and product plans (requirements, architectures, and standards) to keep everything coordinated (Boehm & Turner, 2003). Projects that are highly complex and can span multiple activities will require exhaustive planning of all activities to make coordination between people more dependable. In projects where progress needs to be tracked against a baseline plan, methodologies like RUP are ideally suited. For light methodologies planning is a very ad hoc process done as and when needed.

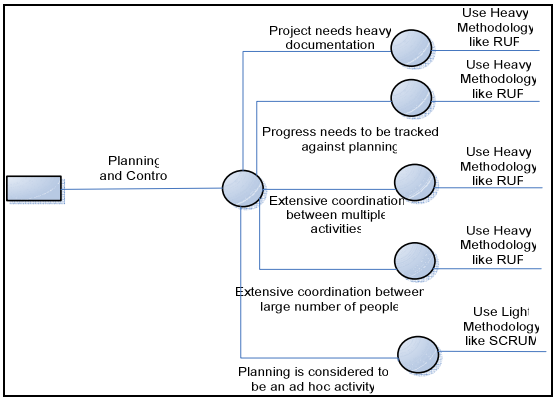

Figure 4: Planning and Control Characteristic Decision Tree

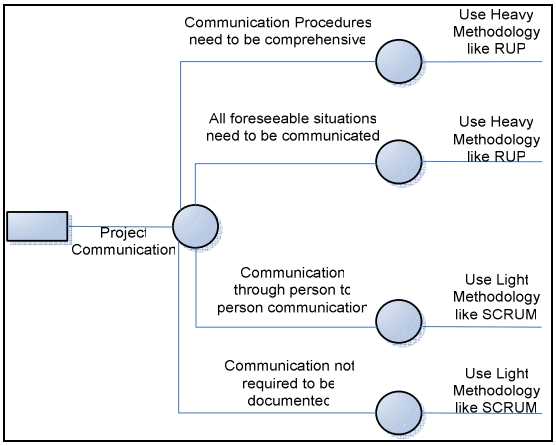

Figure 5: Project Communication Characteristic Decision Tree

4.5 Project Communication Characteristic

Figure 5 indicates when communication procedures need to be comprehensive and need to be documented then the use of heavy methodology makes sense. Light methodologies rely more on person to person communication to pass on relevant information. There is no concept of documenting communication procedures as methodologies like SCRUM rely on tacit, interpersonal knowledge.

4.6 Handling Requirements Characteristic

Figure 6 indicates that when there is a need for formal, complete, traceable requirements it is a good idea to go with a heavy methodology like RUP. RUP identifies and defines requirements and then hands off the software requirements to the appropriate team. RUP also focus more on dealing with quality of nonfunctional requirements such as reliability, throughput, real-time deadline satisfaction, or scalability. Light methodologies express requirements in terms of adjustable, informal stories (Boehm & Turner, 2003). Light methodologies count on their rapid iteration cycles to determine the needed changes in the desired capability and to fix them in the next iteration (Li & Alshayeb, 2001). Customers express their strongest needs and the developers assess what combinations of capabilities are feasible for inclusion in the next developmental iteration (Boehm, 2000).

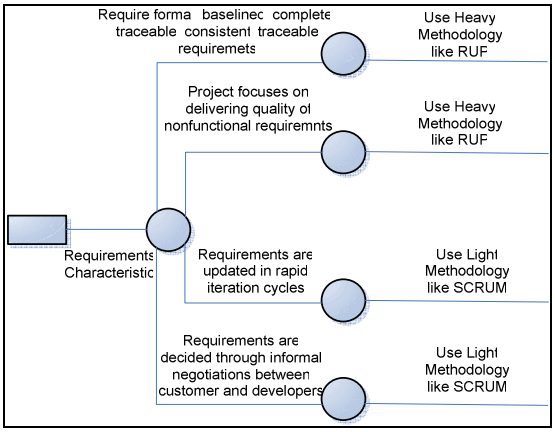

Figure 6: Handling Requirements Characteristics Decision Tree

4.7 Design and Development Characteristic

Figure 7 illustrates that when a project needs to use planning to accommodate foreseeable change a heavy methodology like RUP is a good idea. RUP allows system de-

velopers to reuse software components across multiple product lines. For a project which has to have a robust architecture, RUP is a good fit as predictability and dependability are primary objectives of RUP. Light methodologies advocate simplicity over complexity. Methodologies like SCRUM look at only catering to the minimum requirements as set up by the users or customers.

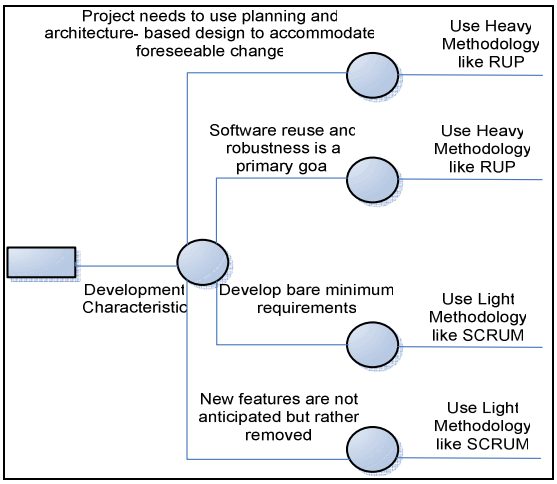

Figure 7: Design and Development Characteristic Decision Tree

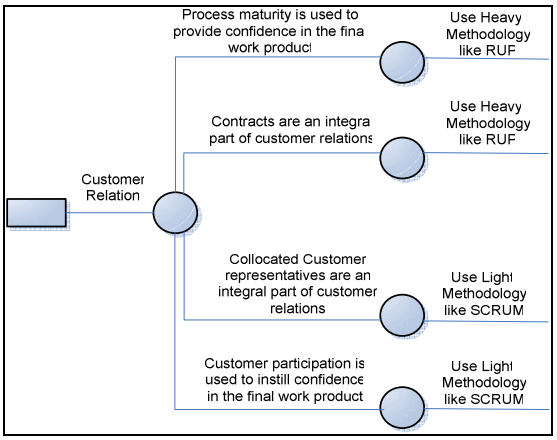

Figure 8: Customer Relations Characteristic Decision Tree

4.8 Customer Relations Characteristic

Figure 8 illustrates that heavy methodologies like Scrum generally depend on some form of contracts between the developers and customers as the basis for customer relations. They try to cope with foreseeable problems by working through them in advance and formalizing the solutions in a documented agreement. Light methodologies strongly depend on dedicated, collocated, customer representatives to keep the projects focused on adding rapid value to the organization (Boehm & Turner, 2003). Light methodologies like Scrum use working software and customer participation to instill trust in their track record, the systems they have developed and the expertise of their people.

4.9 Organizational Culture Characteristic

Figure 9 illustrates that if an organizational culture is very rigid and bureaucratic and demands clear policies and procedures to identify roles for developers and programmers then heavy methodologies like RUP will work well. In RUP every person's task is well-defined and documented. If an organization is characterized by a culture which allows developers a high degree of freedom to define their work roles and policies then light methodologies are a better fit. There is no definite role or scope of work that has to be completed by each person.

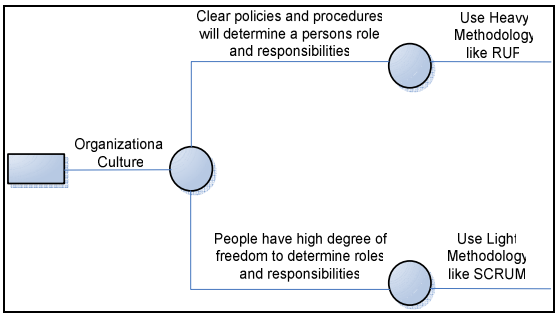

Figure 9: Organizational Culture Characteristic Decision Tree

5. CONCLUSION

Since the emergence of the internet age there has been a considerable change in the approach of project managers with respect to how they address project management issues. Newer methodologies have been developed with the sole intention of being able to adapt to an environment that is ever changing. Unlike their predecessors which followed the classical approach of requirements/design/build these light methodologies follow the mantra of simplicity. With the advent of these methodologies project managers have been faced with a conundrum as to which type of methodologies would be the ideal one to use. As the author has pointed out earlier, there have been numerous ex-

amples of projects failing due to the improper methodology being used.

The study analyzed nine characteristics that define a methodology decision and also studied two popular methodologies (RUP and Scrum), one heavy methodology and one light methodology. Based on these findings, how the light and heavy methodologies each addressed the project characteristics were analyzed. Based on the findings, distinct decision trees for each characteristic were developed. The decision tree showed how the project manager could choose a heavy or a light methodology based on what they wanted to achieve. In cases where the decision trees contradict each other, one pointing to a heavy methodology and one pointing to a light methodology, the nine characteristics must be prioritized based on organizational culture and project situation. For example, the priority assigned to each characteristic for an internet based company in a highly competitive industry on a very important project would be different for a well established fortune 100 company.

The results are not meant to be a substitute for sound project management but as guidelines for project managers and students. A good project manager would look at these results and with the help of other project management tools and techniques would arrive at an informed decision to help manage a successful project.

6. REFERENCES

Beck, K. (1999). Extreme Programming Explained. Boston: Addison-Wesley, pg 157.

Beedle, M & Devos, M, &Sharon, Y. & Schwaber, K. (2000). SCRUM: An extension pattern language for hyper productive software development. Retrieved on September 30th, 2004, from http://jeffsutherland.com/scrum /scrum_plop.pdf

Berinato, S. (2001). The Secret to Software Success. Retrieved September 6th, 2004, from http://www.cio.com/archive /070101/secret.html

Boehm, B. (2000). Unifying Software Engineering and Systems and Systems Engineering. IEEE Computer, pg 114-116.

Boehm, B. (2003). Value-Based Software Engineering. ACM SIGSOFT Software Engineering Notes, Vol. 26-2

Boehm, B., & Turner, R. (2003). Balancing Agility and Discipline: A guide to the Perplexed. Boston: Addison-Wesley

Booch, G. (1995). Object Solutions. Addison-Wesley.

Booch, G., & Jacobson, I. & Rumbaugh, J. (1998). Unified Modeling Language. White Paper. Retrieved on September 20th, 2004 from http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2000/Tp180.PDF.

Brichter, R.N. (1999). The limits of Software, MA: Addison-Wesley

Brown, A. (1996). Component Based Software Engineering. IEEE Computer Society. Pg 140

Cebase.org, (2002). Results from the 3rd e-Workshop on agile development process. Retrieved on 30 September, 2004, from http://fc-md.umd.edu/projects/Agile /3rd-eWorkshop /summary3rdeWorksh.htm

Cockburn, A. (2000). Selecting a Project's Methodology. Retrieved September 24th. 2004 from http://www.eee.metu.edu.tr/~bilgen/Cockburn647.pdf

Cockburn,A.. & Highsmith, J. (2001). Agile Software Development: The People Factor. Computer, pg.131-133. Retrieved September 25th, 2004 from http://www.jimhighsmith.com/articles/IEEEArticle2Final.pdf

Constantine, L. (2001). Methodological Agility: Software Development. Pg 67-69

Crawford, K. (2001). The strategic project office: Business case and implementation strategy. Pennsylvania: PMI Solutions.

Gornik, D (2001). IBM Rational Unified Process. Best Practice for Software Development Teams. Retrieved on September 30th, 2004 from http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/rup_bestpractices.pdf

Highsmith, J. (2002). Agile Software Development Ecosystems. Boston, Addison Wessley.

Hoffman,T. (2003, July 21). Value of Project Management Offices questioned. *Computerworld.* Vol. 37, Iss. 29, pg. 7

Jacobson, I., Christerson, M., & Jonnson, P., & Overgaard, G. (1992). Object Oriented Software Engineering -A Use Case Driven Approach. Addison-Wesley, pg 582.

Jacobson I., & Booch, G., & Rumbaugh, J. (1999). Unified Software Development Process. Addison-Wesley.

Kruchten, P. (1999). Rational Unified Process –An Intorduction. Addison-Wesley.

Li, W. & Alshayeb M., (2001). An Empirical Study of Extreme Programming Process. Proceedings , 17th Intl. COCOMO/Software Cost Modeling Forum. USC-CSE. Retrieved September 20th, 2004, from http://sunset.usc.edu /publications/TECHRPTS/2002

Lim, W. (1998). Managing Software Reuse. Englewood Cliffs, NJ, Prentice-Hall

Pollice, G. (2003). Using the Rational Unified Process for Small Teams: Expanding upon extreme programming. Retreived on September 30th, 2004 from http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/tp183.pdf

Schwaber, K. (1995). Controlled Chaos. Living on the Edge. American Programmer.

Schwaber, K. (1997). Scrum Development Process. Retrrieved September3rd, 2004 from http://jeffsutherland.com/oopsla schwapub.pdf

Schwaber, K., & Beedle M. (2002). Agile Software Development with Scrum. Upper Saddle, NJ, Prentice Hall.

Silwa, C. (2002). Users Warm Up to Agile Programming, Computerworld. Retrieved September 22nd from http://www .computerworld.com/softwaretopics /software/appdev/story /0,10801,69182,00.html

SoftwareMag.com (2004). Project success rates improved over 10 years. Retrieved September 6, 2004, from http:// www.softwaremag.com/L.cfm?Doc= newsletter/2004-01-15/Standish