Process-Driven Software Development: An Approach for the Capstone Sequence

Robert F. Roggio
School of Computing
University of North Florida
Jacksonville, FL 32224
broggio@unf.edu

Abstract

Most computer and information sciences (CIS) programs require a capstone sequence consisting of one or more courses in software development. While the end-product of these project-oriented courses often results in student teams developing and demonstrating some rather impressive applications, it is rare that these products can rival real-world application development that students may encounter in the workplace. This paper thus takes the position it is the process under which an application is developed that is far more valuable to the student than the application itself. Features of several popular heavy weight and light weight methodologies are presented accompanied by methodology recommendations for both one and two course capstone sequences. A decision tree is also included.

Keywords: capstone software development, heavy weight processes, light weight processes, selection of a process.

INTRODUCTION

In many colleges and universities, the capstone sequence has student development teams adopt (or are given) a process to follow. They then proceed to develop a computer-based solution to a problem. In many cases, students may pick and choose a project from a list provided by the instructor; in other cases, projects are assigned. But in almost all cases, student teams both develop an application and demonstrate the application at the conclusion of the course sequence. These presentations are often impressive, as students develop attractive and functional interfaces, accommodate appropriate functionality, and establish a database. Often too, extensive documentation may be required.

The question that this paper addresses is at a coarser level of granularity. It addresses not the 'what' of the application developed (the utility and usability of that interface, the demonstrated functionality via 'customer' testing, and more), but rather the appropriateness of the process used to develop the application. This familiarity with "process" will transcend the slick interfaces, current technologies used, and the database built. The ability to select and adhere to a disciplined, methodological approach to software development, regardless of individual technologies, is a program outcome that we desire in our students. Recognition of different processes each with their suitability to different classes of projects is essential in today's modern development environments.

"Process" can be repeated. Knowledge of basic processes used to support software development is transferable to real life situations Understanding the relative strengths and weaknesses of widely used methodologies and how the characteristics of the project, such as team size, project scope, development constraints, may well influence an appropriate process choice.

The wisdom derived from methodology selection and its impact on the resulting software development effort will significantly add to the experience base of our graduates, regardless of the application ultimately developed in academe'.

Software Development is an expanding discipline, and both new and improved development methodologies seem to emerge each year. The appearance of these methodologies is often the result of attempts to improve existing development processes used to guide the every-increasing complexity and diversity of modern computer-based solutions. Applications today continue to become more and more complicated and require highly tuned skills as compared to applications developed not too many years ago. Customers want more, expect more, and will be only satisfied with more. We need processes that supports these expectations.

Two classes of methodologies have evolved and are, with some modifications, commonly used across many software development industries. They are termed *heavy weight* and *light weight methodologies*. Heavy weight methodologies (please note that I use the term methodology and process interchangeably) are also sometimes called plan-driven methodologies because of features such as comprehensive planning, thorough up front requirements modeling, and typical extensively documented designs, detailed test plans and more. Light weight methodologies, in contrast, are often collectively referred to as agile methodologies, and tend to focus on individuals over processes, working software over documentation, collaboration over negotiation, and responding to change over following a plan. Please note this does not mean that agile processes do not develop requirements or design documents or undertake development of traditional artifacts; rather that the cost in producing these is weighed against other important factors, such as delivering high-quality products earlier and incorporating features into the software that provide clear value to the stakeholders.

This paper will first present a brief description of three heavy weight and three light weight methodologies and their respective features. By observing the features of these methodologies, those charged with methodology selection may become better equipped to select an appropriate process to underpin software development in the capstone sequence.

HEAVY WEIGHT METHODOLOGIES

Heavy-weight methodologies are also known as "traditional" methods; these methodologies are "plan-driven" in that their process may involve business modeling, continue with elicitation and documentation of a complete set of requirements, architectural and detail design, program development, extensive testing, and lastly implementation. Some may be iterative.

Heavy weight methodologies remain the process of choice for many development efforts. These methodologies are well established and often offer senior level management and developers a comfort level that newer, less formal methodologies do not. Note**:** terms heavy-weight, traditional, and plan-driven will be used interchangeably in this paper.

Waterfall Model

According to Reed Sorenson [10], Waterfall is "an approach to development that emphasizes completing one phase of the development before proceeding to the next phase." In Sorenson's article titled, "A Comparison of Software Development Methodologies," he describes which methodologies may be best suited for use in various situations and how the use of traditional software development models is widespread and often regarded as the proper and disciplined approach for the analysis and design of software applications. Each phase comprises a set of activities that should be completed before the next phase can begin.

While the Waterfall approach remains in widespread use today, it is often modified to meet the needs of individual businesses and their tailored way of developing software. The Waterfall approach does indeed continue to offer many advantages where the application to be developed is well understood, unlikely to change appreciably during development, and where, perhaps, the developers may have had previous experience in developing similar applications, and more. However, the methodology does not embrace change, and risk is often addressed late in the development cycle when incurred expenses are at their high point. (Figure 1)

Spiral Methodology

While the Waterfall model has been the basis of software development for many years, its elaborate documentation and rigid adherence to process has often created difficulties and led software practitioners to seek alternative processes. Barry Boehm developed the Spiral Model with its distinguishing feature that with it "…creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process" [4]. The basic tenet of the Spiral Model is that risk is assessed periodically (during each cycle) in the development process thus allowing for frequent project evaluation. (See Figure 2)

The Rational Unified Process (RUP)

"RUP is a process framework that has been refined over the years by Rational Software (and more recently IBM), on a variety of software projects small to large." [9] The RUP approach is considered by many to be a lighter heavy weight method, where the aim is to work with short, time-boxed iterations within clearly articulated phases. Elaborate workflows, specification of activities and roles characterize the RUP. The RUP was not originally intended to be a heavy-weight process, but its wide adaptation – particularly by many very conversant with the waterfall model – has resulted in a more rigid methodology than originally intended by its authors. The current version of the RUP has a number of significant tools to assist in tailoring the RUP to individual projects and contains a comprehensive suite of support tools. (See Figure 3)

The RUP is defined to be a use-case driven, architecture-centric, iterative development process. Thus while it is considered a lighter heavy-weight process, the RUP's adherence to iterative development, extensive use of use-cases, high levels of customer involvement, and a clear approach to embracing changes during the development process makes this process more modern and to some degree (most feel) that it is somewhat 'lighter'.

AGILE METHODOLOGIES

The agile methods place more emphasis on people, interactions, working software, customer collaboration, and change, rather than on process details, workflows, contracts and plans. Agile methodologies continue to gain great popularity in industry although they compromise a mix of accepted and sometimes controversial software development practices. Although plan-driven, heavy weight approaches are still largely used for larger projects that require the ultimate quality often in very critical systems, in many situations more significant growth lies with agile or flexible methods, as customers demand rapid delivery, more developer contact and interactions, and often want to have their fingers on the pulse of development as they may continue to introduce change on a more somewhat regular basis.

Feature Driven Development

Feature Driven Development (FDD) is a model-driven, short-iteration software development process. [1] 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. According to Boehm and Turner, FDD consists of five processes: develop an overall model, build a features list, plan by feature, design by feature, and build by feature [3].

The FDD methodology produces very frequent and tangible results. Small blocks of functionality that have specific user value are delivered. This popular development approach provides for very effective progress tracking; the overall application evolves as features are added and deployed.

Scrum

Scrum is an iterative software development approach which aims to deliver as much quality software as possible within a series of short-time boxes called "Sprints" [11]. Sprint begins with a Sprint Planning Meeting where the product owner, customers, developers and other relevant stakeholders meet to define the immediate Sprint Goal. This immediate sprint goal is selected from a Product Backlog, which is a list of requirements. (See Figure 5) The product owner is required to prioritize these requirements.

After addressing the Product Backlog, the Scrum development process focuses on addressing a Sprint Backlog. According to Linda Rising and Norman Janoff [8], the Sprint Backlog is the final list of product items transferred from the Product Backlog. A scrum team breaks down the Sprint Backlog into small tasks and allocates them to its team members. The Sprint Backlog is updated daily to reflect the current state of the Sprint

Figure 5 also more clearly illustrates some of the management aspects of scrum software development; its thirty day iterations and the daily scrum meetings. With a thirty day iterative cycle, risk and change can be continually reassessed and readily managed for each iteration.

eXtreme Programming (XP)

XP is "a discipline of software development based on values of simplicity, communication and feedback" [14]. This methodology works by bringing the whole development team together to collaborate in simple practices, with enough feedback to allow the team to see and monitor their project improvement and be able to address any issue that occurs throughout the phases of the development process.

Almost any software development effort will experience requirement changes before it is completed. Agile methodologies such as Extreme Programming seem to be the most suitable for responding to requirement changes. Communication between team members and the customer is conducted through informal contact via face-to-face meetings. The customer is part of the team. Obviously, this type of communication is an advantage for both parties; customers enjoy being partners in the software process, and the development team has ready access to the customer for questions and continuous feedback.

XP espouses simplicity in every undertaking. In particular, applications are developed using a simple design, small releases, continually restructuring components for better performance and more. Testing is extensive and focused on the latest component. This is facilitated by XP's adherence to the pair programming concept, where pairs of professionals 'own' their code, so to speak.

XP develops software incrementally, but methodically using short time periods measured in weeks rather than months. High quality software occurs as software is continually refined and improved during the iterative cycles. This methodology attempts to avoid activities and artifact production that do not have a clear value that directly supports the immediate (or near immediate) goal.

XP has twelve core practices as can be found in [12]. Strong adherents to XP claim that all twelve must be implemented to gain the maximum value from this process.

CAPSTONE SEQUENCE

So, how do does this information assist us in capstone course development? How can we use these categories with some sample methodologies to, perhaps, drill down to a methodology that best fits our capstone sequence? While there will be no exact fit of a methodology to project characteristics, the text ahead discusses a series of questions that might assist in this decision. Such a series of questions and answers may be best illustrated not only in text, but also a decision tree.

The first look is at the characteristics of a project. While a comprehensive look at project characteristics for all projects is clearly well-beyond the scope of this paper, a number of commonly used characteristics may be used within the context of an academic setting. The project characteristics considered below include length of capstone sequence (one or two sessions), degree of documentation planning and control desired, project communications with customer / sponsor, and the design and development environment. These were arbitrarily selected, but do appear to answer a number of important questions that might bear on the problem.

It is important to note that there are many other very significant parameters in real world project characteristics that do not bear on the academic setting. In the workplace, team size is significant, as generally larger teams often use heavy-weight methodologies. Other workplace parameters? Consider team skills - large teams can absorb less experienced professionals more readily than light weight approaches. The academic setting has little latitude here. Experienced, senior developers can nurture younger individuals in the workplace. Testing - the many faces of tests from a variety of internally-undertaken testing to external testing approaches. Not likely in an academic setting. Customer support – insuring adequate training to those responsible to customers for problem identification. Training – actively training users in the particulars of the application, and more. The academic setting is constrained by time (academic session), experience of the team members, methodology used, documentation required, planning, and so much more.

First Level Decision – Heavy Weight or Light Weight Methodology

Length of Capstone Sequence; one or two terms: The first consideration is perhaps the most significant. Is the capstone sequence a single course or is it two courses? If the capstone sequence is two semesters, then there is more time for methodologies that have more detailed sets of required activities and artifacts produced. While this is very simplistic to offer because there are so many other parameters to be considered, generally, with more time, a heavy-weight methodology might be favored in in this instance. Both the water fall and the spiral models (particularly if risk is emphasized) might be selected. If an iterative approach that provides for short development cycles each with change, testing and assessment is desired, the RUP might be a wise choice. The RUP subscribes to many modern programming practices, and while normally considered somewhat heavy, it does support change, risk, iterative development, use case analysis / design and more. See Figure 6.

Documentation, Planning, and Control: If the sequence is to require detailed manuals produced by student teams, considerable time must be provided to produce and validate these artifacts. Typical documentation might include user manuals, maintenance manuals, and operations manuals. If the extent of documentation is significant, a heavy-weight methodology is likely better. Similarly, if detailed plans, tracking activities against tasks, strict timelines and monitoring are required, this too may legislate toward a heavier methodology If lesser or perhaps just on-line help is required, then a light-weight methodology might be considered.

Project Communications: If the customer (presumably the instructor, an agency on campus, or perhaps a local business) is a key part of the development team and is readily available for contact throughout the development process for verifying various intermediate results, and consultation in general, then a light-weight methodology may be preferred. This may often be the case in an academic setting. But if the requirements are provided up front with limited chance or periodic times for discussion or detailed interaction with the customer, then a heavy-weight methodology is preferred.

Design and Development Environment: If a robust, well-designed and coded product is desired that is explicitly designed for extension and reusability within a carefully orchestrated and documented architecture, one might consider a heavy weight methodology. If an architecture and design are required to assist the tasks at hand and are then developed to drill down within that software architecture and/or organization at certain points during development, then a light-weight approach is preferred. If testing a specific set of features (along with some regression testing, of course) as features are incrementally added to an evolving application are emphasized, then a light-weight approach appears to be better.

Second-Level Decision – Methodology Selection (See Figure 7)

Consider the Heavy-Weights: Neither the Waterfall nor the Spiral methodologies are terribly iterative in nature, even though some implementations of these methodologies do provide for limited feedback and retrenching. The RUP, however, is iterative by design. Of these three methodologies, the RUP is the most iterative.

Change is not well supported in the Waterfall model and addressed via the cyclic nature in the Spiral model. The RUP, due to its iterative and incremental nature espouses the embracing of change.

Consideration of risk is delayed in the traditional waterfall model, is visited cyclically in the spiral model, and is addressed up front in the RUP approach. Yet, if risk is anticipated to be minimal, that is, the development environment appears to remain relatively stable as do requirements, a heavy-weight approach might be preferred.

All of these processes require considerable documentation and well-conceived plans with task assignment, tracking and workflows. Light weight methodologies approach these 'roles' (as found in the RUP) much less formally and expect individual development team members to do when needs are identified.

Consider the Light-Weights: Feature-driven development (FDD), Scrum, and XP were briefly presented. . FDD is a model-driven short-iteration development process. With its short (typically) two+ week design by feature build by feature iterations, rapid, incremental development may occur once an initial plan is built.

This is a simple approach that concentrates on a plan, initial set of identified features, followed by iterations of design. code, test, deliver iterations. With a short iterative cycle particularly in a one session capstone course, FDD might be a consideration.

With its customer-supplied prioritized set of requirements which is broken into smaller features that are assigned to sprints, Scrum may be the methodology of choice. While much of the formality of the heavy-weight processes is eschewed, 30-day iterations selected from a product backlog are undertaken. With daily sprint meetings where problems are surfaced and resolved, this might be difficult in an academic setting. But the adherence to sprint meetings where everything of interest is surfaced and addressed offers a great chance for students to interact by speaking and communicating with each other and the customer.

XP is often considered the 'lightest' of the light-weight methodologies and perhaps the most controversial. XP espouses simplicity in everything that it does, it seems, and this is often criticized by traditional software developers. This methodology works by bringing the whole team together to collaborate on simple well-defined practices, with enough feedback to allow the team to see and monitor their project improvement and be able to address issues that occur throughout the phases of the development process. Basic XP tenets, such as pair programming, collective ownership of products, the entire team sitting together in one room, integration many times a day, and the philosophy to merely build just enough to meet today's requirements are among XP principles. If XP is to be used in a capstone sequence, its basic tenets and organization must be carefully considered. While the way XP does business seems to favor a one session capstone sequence, the ability of students to work together in a larger setting coupled with typical little real-world development experience might be causes of concern if this methodology is selected.

Scrum, XP, and agile processes in general have strong real-world adherents many of whom are widely acclaimed. Providing students with such an environment might have significant value. It is significant, however, that many agile features are often vehemently challenged by proponents of more traditional heavy-weight methodologies. It may also be quite difficult to enforce the development discipline necessary for such a process in academe'.

CONCLUSIONS

Simplistic Approach

It is important to restate that the decision tree approach taken is both very simplistic and likely incomplete. Figures 6 and 7 do not adequately address other parameters essential from an academic point of view. For example, class size and team size were not considered, although a nominal size of from three to five students was assumed. But in truth, team size can impact the expected course outcomes. Similarly, instructor-sponsored projects may be more readily accommodated within a single semester, while client-sponsored projects, by their very nature, may require a two-semester sequence. Client-sponsored projects may require very specific documentation. Frequent communications between the client and the development team may be prescribed and thus require more time – regardless of methodology.

Essential to Undergraduate Experience

Regardless, the capstone sequence is designed to culminate the undergraduate experience. Business modeling, requirements capture and modeling, architectural and detail design, program development, comprehensive testing, and customer implementation take place. Soft skills are brought to bear in documenting and presenting various artifacts of the application. Yet the selection of a proper development methodology appropriate to the nature, objectives, and expected outcomes of the capstone project course constrained by the realities of an academic environment must be carefully undertaken. It is woefully insufficient to hand out requirements and to tell student teams to build software to accommodate these requirements.

The experience gained by the student in adhering to some kind of disciplined process far exceeds the value of the application developed itself. It is the understanding that software development is not simply sitting down and writing code, but rather a painstaking, complex, multifaceted undertaking involving people, procedures, and process. The capstone sequence provides an unmistakable and an irreplaceable learning outcome essential for today's computing professional that must be realized.

Whether the capstone sequence is a single session or multiple sessions, care must be taken in determining the underlying process necessary to support the software development effort with its many academic and real world constraints/

REFERENCES

[1] Abrahamsson, P., O. Salo, J. Ronkainen and J. Warsta, Agile Software Development Methods: Review and Analysis, Julkaisija Utgivare Publisher, Finland, 2002.

[2] Beck, K., Extreme Programming Explained, Addison-Wesley, Boston, 1999, p. 157.

[3] Boehm, B. and Tuner, R., Balancing Agility and Discipline, Pearson Education, Inc., Boston, 2004.

[4] Boehm, B, "A Spiral Model of Software Development and Enhancement," ACM SIGSOFT Software Engineering Notes 11, 4 (1998), pp. 14-24.

[5] Coram, M. and S. Bohner, "The Impact of Agile Methods on Software Project Management," Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops, 2005, pp. 363-370.

[6] Kruchten, P., The Rational Unified Process: An Introduction, Addison-Wesley, Boston, 2004, p. 43, 81.

[7] Nerur, S., R. Mahapatra and G. Mangalaraj, "Challenges of Migrating to Agile Methodologies," Communications of the ACM 48, 5 (2005), pp. 72-78.

[8] Rising, L. and N. Janoff, "The Scrum

Software Development Process for Small Teams," IEEE Software 17, 4 (2000), pp. 26-32.

[9] Smith, J., "A Comparison of RUP® and XP", www.yoopeedoo.com/upedu/references/papers/pdf/rupcompxp.pdf

[10] Sorensen, R., "A Comparison of Software Development Methodologies," www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1995/01/Comparis.asp

[11] Sutherlan, J., "Future of Scrum: Parallel Pipelining of Sprints in Complex Projects" agile2005.org/RP10.pdf

[12] Sambasivam, Ganesh, "Extreme Programming (XP)", www.agilealliance.com/articles/ganeshambasivamextre/file, last revision August 2004.

[13] Control Chaos, www.controlchaos.com/about/

Appendices

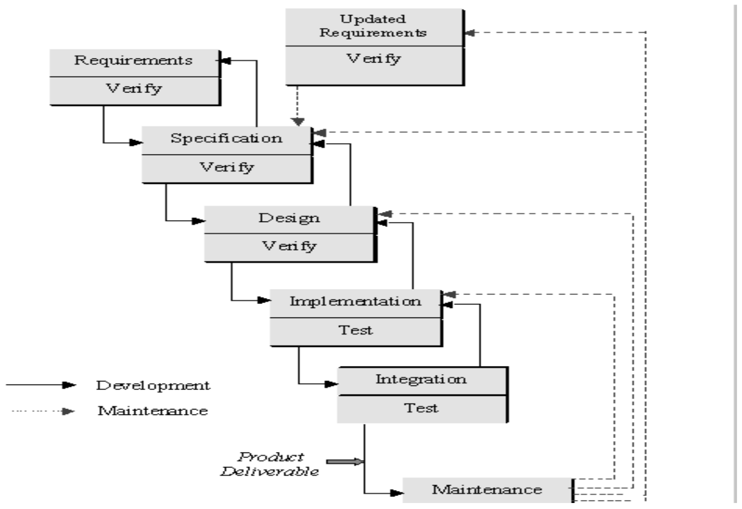

Figure 1: The Traditional Water Fall Model

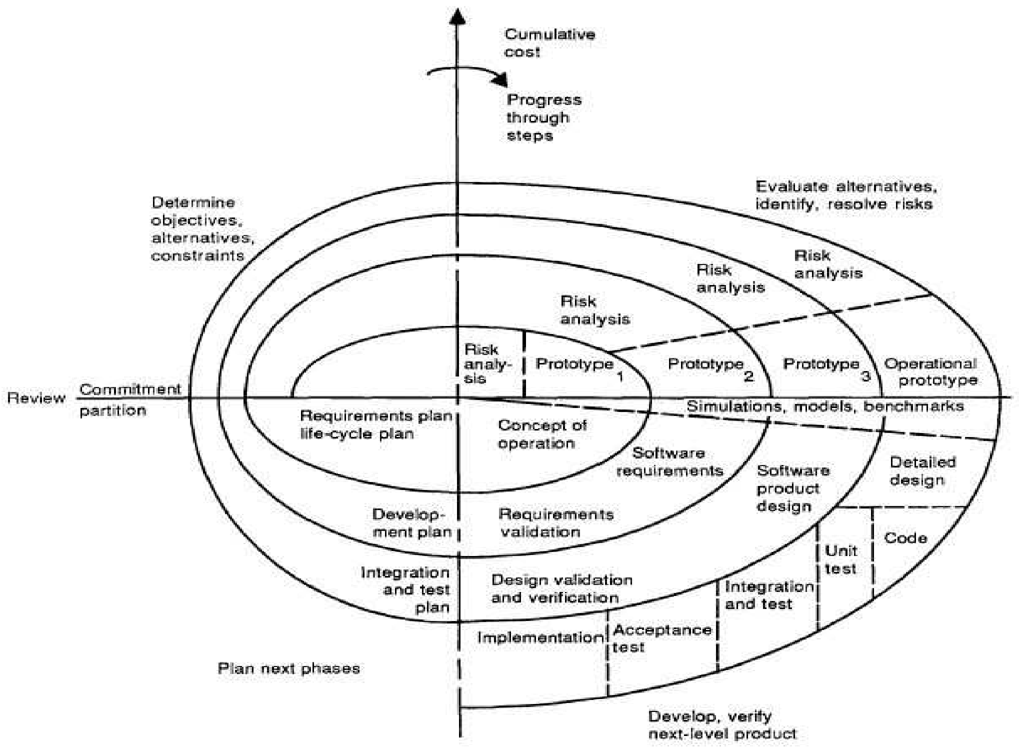

Figure 2: Spiral Model of Software Process [4]

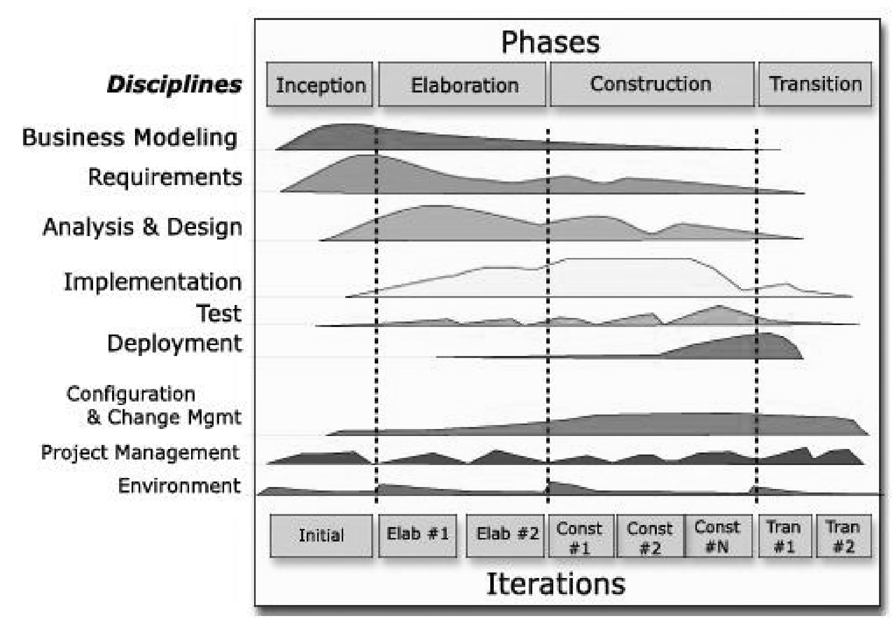

Figure 3: RUP's Four Phases and Nine Disciplines [9]

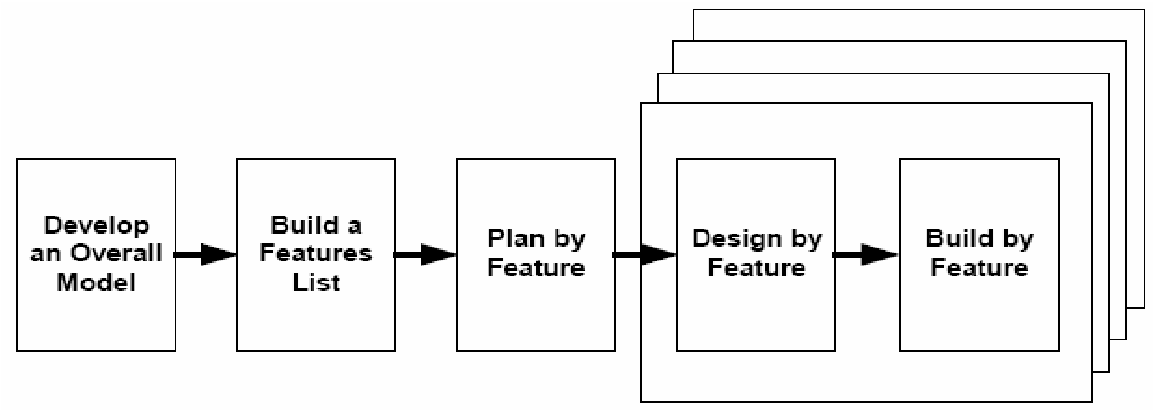

Figure 4: FDD Process (Adapted from [1]

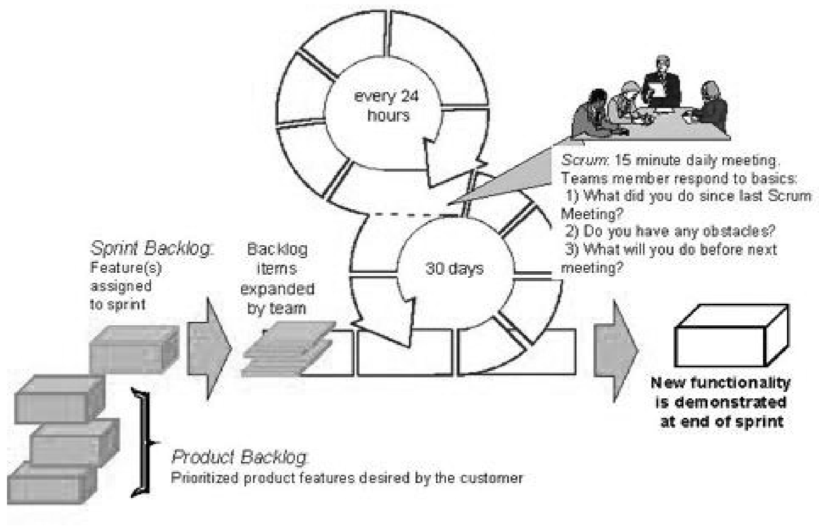
every 24 hours
Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
30 days
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Figure 5: Scrum Process Flow [13]

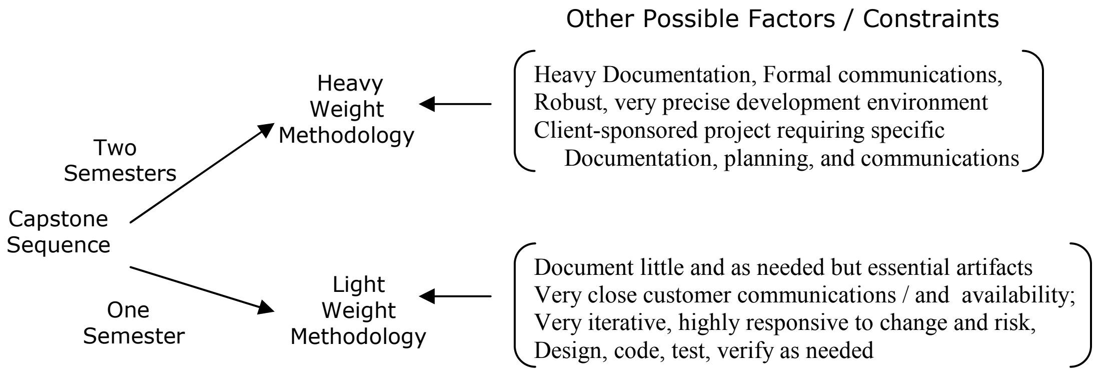
Other Possible Factors / Constraints
Capstone Sequence
Two Semesters
One Semester
Heavy Weight Methodology
Light Weight Methodology
Heavy Documentation, Formal communications,
Robust, very precise development environment
Client-sponsored project requiring specific Documentation, planning, and communications
Document little and as needed but essential artifacts
Very close customer communications / and availability;
Very iterative, highly responsive to change and risk,
Design, code, test, verify as needed

Figure 6: First Level Decision Tree. Heavy or Light Weight Methodology

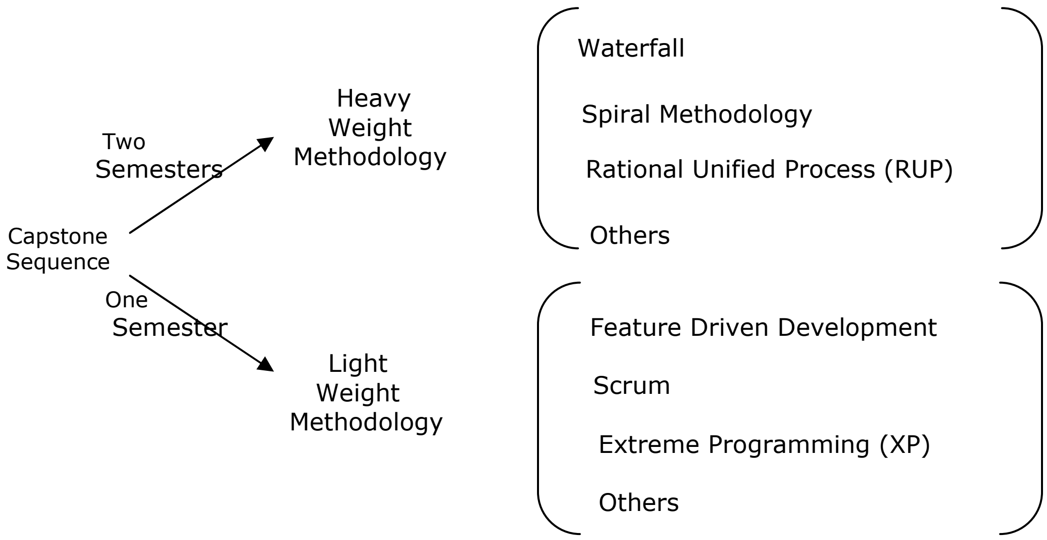

Figure 7: Second Level Decision Tree - Methodology Selection