

SOFTWARE DEVELOPMENT LIFE CYCLE METHODOLOGY SELECTION FOR
DEVELOPMENT AND MAINTENANCE

A THESIS

SUBMITTED ON EIGHTH OF JANUARY, 2012

TO THE DEPARTMENT OF SOFTWARE ENGINEERING

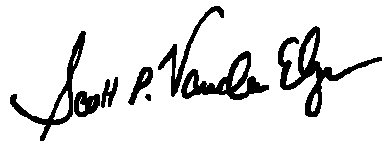
OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN

SOFTWARE ENGINEERING

BY



Scott P. Vanden Elzen

APPROVALS

Douglas Hart, Thesis Advisor

Ranked Faculty Name

Ranked Faculty Name

Abstract

The Software Development Lifecycle Methodologies (SDLC) that organizations use for project creation may not be the best selection for project maintenance. There are frameworks that help compare various SDLCs by evaluating project complexity, uncertainty, and required level of quality, and methodology phase. This project extends the CuQuP SDLC selection framework by Yusof, Shukur, and Abdullah (2011) to add support for Agile SDLCs. Six organizations were surveyed asking about the SDLCs they used and the criteria used to select them. The updated CuQuP framework was then used to quantitatively score various SDLCs for all six organizations. It showed that most organizations selected the best methodology for their organizations project creation and maintenance. However, there were other organizations where a different methodology for project creation versus maintenance would, in some circumstances, provide better results.

Acknowledgements

I would like to thank my wonderful wife Michelle for her love and support during my Graduate school studies. I am also thankful to the managers who participated in the survey, whom without; this thesis would not have been possible.

Table of Contents

List of Figures	v
List of Tables	vi
Section 1 – Overview and Content	1
Statement of Problem.....	1
Purpose Statement.....	1
Rationale	2
Research Questions and Hypotheses	3
Section 2 – Review of Literature and Research.....	4
Literature Review Conclusion	13
Section 3 – Methodology	15
Participants.....	15
Data Analysis	15
References.....	21
Appendix A. Questionnaire	23
Appendix B. Annotated Bibliography	25

List of Figures

<i>Figure 1</i> Classic steps of a software methodology	5
<i>Figure 2.</i> Contingency Approach	6
<i>Figure 3.</i> Houston matrix quadrant assessment.....	8
<i>Figure 4</i> Scope of methodologies	10
<i>Figure 5.</i> Decision making process	12

List of Tables

Table 1. <i>Criteria used in SDLC selection / modification</i>	16
Table 2. <i>Recommended SDLC</i>	18

Section 1– Overview and Content

The way software is developed is as important as the language it is written in or the computer it runs on. It is actually more important because many software projects outlive their original language or target computer system. As long as the original business need remains, some kind of software must exist to fill that need. Today a team can organize to build software in many ways. Software development lifecycles (SDLC) have been defined for more than 20 years to help describe the processes, procedures, and artifacts a team uses. Some example methodologies are SCRUM, Waterfall, and Kanban. All of these methodologies have places where they work well and where they do not. Factors that go into selecting a SDLC include team location, team size, risk of errors or omissions, time to market, the number of changes expected, how long the project will last, and what kind of updates or maintenance releases will be required.

Statement of Problem

The methodologies teams use for their maintenance releases are as important as those used to create the product for its initial release. I will review if the SDLC used for project maintenance was the same for project creation, and if the attributes of the SDLC used for project creation were consistent with the needs team during project maintenance. I will review six different organizations, methodologies they used, and why they used the SDLC chosen for project creation and project maintenance.

Purpose Statement

The goal of this study is to create a list of criteria that should be considered when selecting a SDLC for both software development and software maintenance. The criteria will take into consideration organization size, the type of product, timeframes from development to release, team location, and team experience.

Rationale

Companies spend billions of dollars on software each year. The Standish Group (1995) reported that companies spent over \$250 billion on application development for approximately 175,000 projects. With the rate at which companies continue to use software to solve business needs, this figure certainly has increased since then.

The software development lifecycle (SDLC) methodology has a determination in the length of time it takes to develop the software and the number of defects found in the software. For instance, Shine Technologies (2003) reported that companies using Agile methodologies showed a 88% improvement in productivity, 84% improvement of software quality, and 49% reduction in costs (Shine Technologies, 2003). Clearly selecting the right methodologies has a significant impact on software quality and costs. Currently, the SDLC used for project maintenance is the same as used in the development process, which may not fit the needs for the maintenance process, or worse, there is no formal process at all (April & Abran, 2008). Other SDLCs or tools maybe helpful to reduce problems and improve the user experience (Sarkar, Sindgatta, & Pooloth, 2008).

Building quality software is a difficult challenge. So it is important for an organization to select a defined SDLC for their projects. Customers, developers, analysts, testers all need to agree on what should be built, in what order, and how it should be built. Having a formal process increases quality and reduces risk. Fitzgerald (1998) put forward several arguments for organizations to use an SDLC.

- Methodologies help cope with the complexity of software development
- Methodologies reduce risk and uncertainty by creating transparency
- Methodologies can provide a library of techniques and resources

- A formal methodology supports certifications like ISO or CMM.
- Some governments or institutions require particular methodologies be used for their projects.

However, even if a management team wants to make an informed decision on which SDLC to use, it is not an easy task. A team of key stakeholders with different roles within the organization should be formed to help evaluate the different candidate methodologies. The team should understand how to choose a framework to evaluate the methodologies. Gruner, Klopper, and Kourie (2007) evaluated frameworks put forward by several authors. They found choosing a framework that allowed for quantitative comparison of different methodologies made it easier to choose the appropriate SDLC methodology for the project. Helping organizations determine when to use the appropriate SDLC will help managers and developers provide better processes to their company and improve the total software experiences for their users.

Research Questions and Hypotheses

- Was the software development lifecycle (SDLC) methodology used by companies for software maintenance specifically selected by the management team for their maintenance cycles?
- What criteria, if any, were used by the management team to select the SDLC?
- Do companies differentiate between software development and software maintenance during their development cycles?
- What kind of research and resources do companies use when making a selection of an SDLC?
- Do they use a decision support framework to help make the selection of a SDLC?
- Is the environment taken into account when selecting a SDLC?

Section 2 – Review of Literature

The research reviewed here is ordered chronologically. The author hopes to guide the reader through the history of software development methodology research. The bulk of research in this area is recent, with only a handful of articles predating 1980. Although a fair amount of scholarly material has been published in the area, it is not available to mainstream information technology professionals, partly because of the immaturity of the field.

Understanding the basic concepts of software development methodologies is necessary to be able to evaluate the best software development lifecycle (SDLC) methodology. Gruner et al. (2007) explained that all software projects go through the phases of requirements gathering, business analysis, system design, implementation, and quality assurance testing. As shown in Figure 1, each loop feeds back information and tasks to the previous steps.

“Construction of quality software is not an easy activity” (Gruner et al., 2007, p.57). Using a formal methodology for developing the software reduces risk, increases quality, improves visibility, makes project management easier, and provides a standardization that allows more than one developer to work on a project. Conversely, developing software without a formal methodology increases the chance of failure of the team and the project (Fitzgerald, 1998).

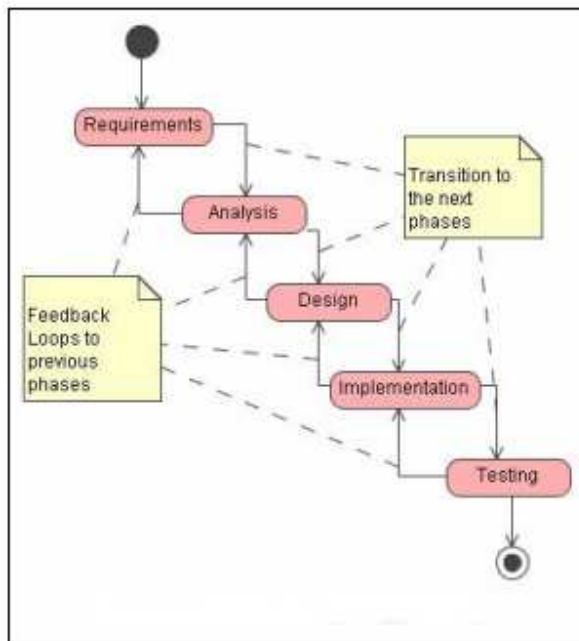


Figure 1. Classic Steps of a Software Methodology. From “Assessment of a framework to compare software development methodologies” by S. Gruner, R. Klopper, & D. Kourie, 2007. the Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information, 56-65.

Burns and Dennis (1985) introduced the idea of selecting an SDLC methodology based upon the type of project. They only had three SDLCs to choose from: System Life Cycle (essentially waterfall), prototyping, and a mix of the two. However, they introduced some key concepts in the selection process. The selection should be made with a combination of project complexity and project uncertainty. They defined project uncertainty as a combination of the factors determining the project’s structure, user task comprehension, and developer task proficiency. Project complexity is determined by the project size, number of users, volume of new information the system creates, and complexity of the system itself. Burns and Dennis used the ranking of these factors to place each project in a matrix, as shown in Figure 2.

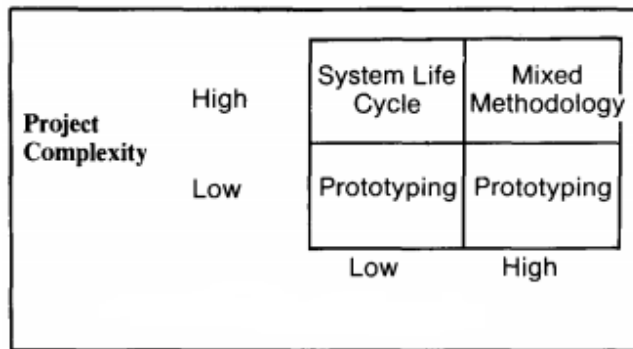


Figure 2 Contingency approach. From “Selecting the appropriate application development methodology.: by R. Burns, & A. Dennis, 1985, *SIGMIS Database 17*, 19-23.

Cockburn (2000) defined the *Big-Methodology* and put forward the following principles: “A larger group needs a larger methodology” (p.65); “A more critical the system – one whose undetected defects will produce more damage – needs more publicly visible correctness” (p. 65) that needs to go into its construction; a small increase in methodology size or density increases cost; face to face interaction is the most effective form of communication; project priorities (time, quality, or visibility) and the methodology of the designer. He continued to use these factors to create a risk ranking, which is dependent upon the factors above and the number of people working on the project. Cockburn brought forward the factors to be considered, but did not explain how to select a particular methodology.

Little (2005) presented prescriptive recommendations depending upon the type of project. He defined a system that calculates project complexity using a 1, 3, 5, 7, 10 scale for mission criticality, collocation of the team, capacity, domain expertise, and dependencies. It calculated uncertainty using the same scoring range for uncertainty, project duration, and dependencies. Using those complexity and uncertainty each project is assigned a type as shown in Figure 3.

For any type of project, Little (2005) had set a set of core process practices:

- An aggregate product plan (target date, product vision, and a list of high-level features that must be completed, target market and supported platforms).

- A prioritized set of features that articulate the must haves, should haves, might have, and cannot haves of the project.
- A level of quality agreement with all stakeholders.
- Continuous integration so the entire project is automatically built at least nightly.
- Involvement of expert users in the development team as developers, testers, or business analysts.
- Having a weekly project dashboard available to stakeholders to view project status. It includes the project plan, quality metrics, current risks, and any changes to the release estimate.

To these core processes, Little (2005) added additional work practices depending upon the type of project as shown in Figure 3.

- Dog projects typically only need the core process practices.
- To colts he added short iterations, daily-stand-ups, and automated unit tests.
- For cows he added more rigorous requirements management, specifications of interface definitions, and detailed project plans with the critical path and any subprojects identified.
- Bulls are the highest risk projects because of their large size and complexity, and they get all the items added for colts and cows along with the most seasoned project managers to insure the success of the project.

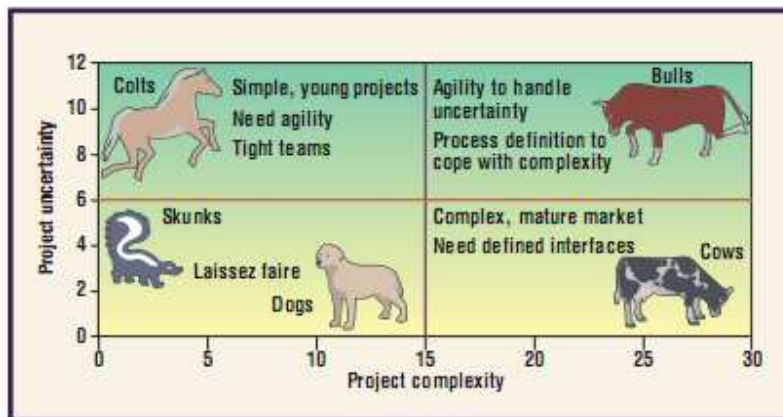


Figure 3. Houston matrix quadrant assessment.. From “Context-adaptive agility: Managing complexity and uncertainty”, 2005, *IEEE Software* 22, 28-35.

Avison & Fitzgerald (2006) did a comprehensive job providing criteria that might be considered in comparing methodologies. They included:

rules, total coverage, understanding the information resource, documentation standards, separation of logical and physical design, validity of design, early change, inter-stage communication, effective problem analysis, planning and control, performance evaluation, increased productivity, improved quality, visibility of the product, teachable, information systems boundary, designing for change, effective communication, simplicity, ongoing relevance, automated development aids, consideration of user goals and objectives, participation, relevance to practitioner, relevance to application, integration of technical and non-technical systems, scan for opportunity, separation of analysis and design. (p. 591)

Any one methodology cannot encompass all of these factors. Instead, these items should form the basis of a checklist when evaluating methodologies. Some criteria may be important and others ignored depending up on the application.

Avison & Fitzgerald (2006) created a framework for comparing methodologies first by breaking down the problem into seven elements.

1. Philosophy – Is the methodology system oriented, data oriented, or people oriented?
It is important that the philosophy of the methodology selected match the organization and team implementing it.
2. Model – “The model is an abstraction and representation of the important factors of the information system or the organization” (p. 602). It is used to capture the essence of a problem and facilitates communication.
3. Techniques and Tools – what techniques, tools, artifacts does the methodology require or recommend.
4. Scope – What phases of the lifecycle does the methodology cover? (See Figure 4).
Projects typically start as part of a corporate strategy and go through several steps until they reach a maintenance phase. While Avison & Fitzgerald discussed Agile methodologies in their book briefly, they did not include them in the scope comparison and invited the reader to do so. I extended phase comparisons for generic Agile methodologies, Scrum, and Kanban.
5. Outputs – What does the methodology produce at each stage and at project completion? Are the artifacts to be delivered required or recommended?
6. Practice – How difficult is it to implement the methodology? What kind of support and training are needed?

7. Product – What is included with the methodology? Is it a software product to be used by the development team, a set of documentation manuals, or a website?

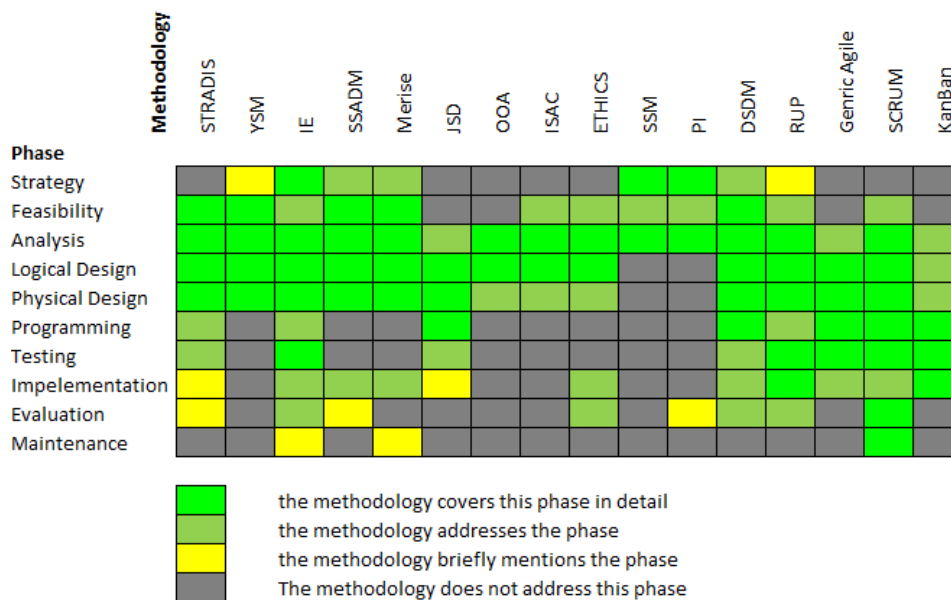


Figure 4 Scope of methodologies

From Avison & Fitzgerald’s perspective, comparison of methodologies involved many subjective measures that depended on the philosophy of the team and not really tallying up a checklist.

Gruner et al. (2007) proposed a multipart process to selecting a methodology shown in Figure 5. First, a multidisciplinary team is formed to do the evaluation. Members of the evaluation team need to include members of management, the development team, the quality team, and any other important stakeholders. Then, a framework is selected to use as an evaluation tool to compare different methodologies. The framework should allow quantitative comparison to reduce the subjective bias of the team.

Gruner et al. (2007) also suggested the framework itself needed to be adjusted to account for organizational requirements that might be different or unusual. The specific requirements for the project are gathered and the team adjusts the framework. Specific requirements included:

- “Facilitate the communication and understanding between the business and IT divisions”
- “Create a library of business processes” that are reusable from project to project
- Create artifacts of functional documentation and business documentation so the project can be supported in the future.
- Create a common vocabulary between users and developers.
- Create a collaborative environment for users and developers.
- Facilitate business process re-engineering when appropriate
- “Integrate with existing processes such as project management, quality assurance, and change control”.
- Create a repeatable process.

The framework and requirements are used to select a short list of possible methodologies by comparing the requirements to the SDLC’s underlying principles. SDLCs whose principles are not congruent with the organizational requirements are excluded. The framework is then applied to the different candidate methodologies, and scores are compared based the raw score and weighting of the various framework line items. The team can compare the scores along with other qualitative components to make a decision on which SDLC best fits the needs of the project (Gruner et al., 2007).

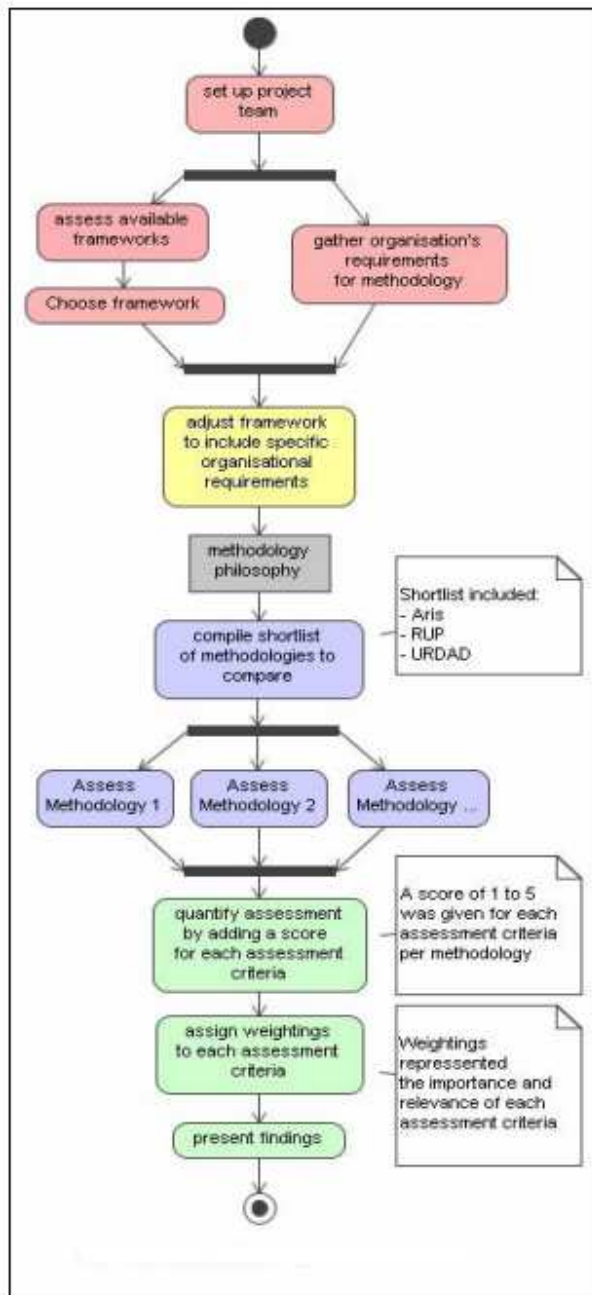


Figure 5. Decision making process. From “Assessment of a framework to compare software development methodologies”, by Gruner et al.,2007. *Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information*, 56-65.

Yusof, Shukar and Abdullah (2011) proposed a methodology for selecting a software development life cycle (SDLC) methodology they named CuQuP. Their approach is a synthesis of the work done by Burns and Dennis (1985); Cropley, Yi, and Cook (2003); Little (2005) and

Avison and Fitzgerald (2006). They produced a weighted scheme based upon three main elements: project complexity and uncertainty level; system quality criteria; and the scope of the methodology phase. They then used this scale against every methodology being reviewed. The methodologies that ranked the highest deserve further consideration.

Yusof et al. (2011) used Little's (2005) the research to determine the level of system complexity on a scale of 1-100. Organization size, capacity and competency of the project participants, location of the participants, critical level of system, system process and integration, user requirements, number of users, evolution, data requirements and process" (p. 3) were considered. They also used Little's 1-100 ranking of uncertainty based upon market uncertainty, technical uncertainty, project duration, and dependencies as weighting factors.

Finally, Yusof et al. (2011) used Cropely et al.'s (2003) the research that employed system quality criteria and strategy, feasibility, analysis, logical design, physical design, programming, testing, implementation, evaluation, and maintenance. The scores for these criteria were weighted and then each methodology was evaluated using the three scales, and a cumulative score was created for ranking. Yusof et al. (2011) did not take into account the difference between initial project creation and the continued maintenance of an existing software project.

Literature Review Conclusion

As a whole, the literature does a complete job taking into account the various aspects of a project including complexity, uncertainty, and system quality requirements. The authors presented a number of different ways to select a SDLC. Some were more formal and used a decision support framework interwoven with organization requirements; others took a simpler approach and looked only at the project complexity, project duration, and risk to make a

decision. Although none of the methodologies specifically addressed software maintenance projects as compared to standard development projects, they could be adapted by incorporating the organizational requirements.

Section 3 – Survey Findings

Participants

I conducted in person interviews with seven managers or software development management consultants from different software development companies in northeast Wisconsin. They represented a cross section of companies: (a) companies that develop software for internal systems used by the company, (b) companies that do custom development for their clients for a variety of projects, and (c) companies that create a software product to sell to multiple clients. There were both managers who had deep experience and had used several different SDLCs, and some who were younger and were familiar only with currently popular SDLCs. I used a standard questionnaire as found in Appendix A as the basis for the interview. During the interview process, I learned additional and supporting information, which deepened my understanding of the individuals and their companies.

Data Analysis

Not surprisingly, most companies do not change their development methodology for different projects or for project maintenance as shown in Table 1. Most of them had little or no formal methodology for critical maintenance upgrades (break-fix updates). This likely due to three factors: (a) The organizations do not have projects that are significantly different from each other, so going through the selection process each time yields little value for the cost; (b) Organizations build an organizational process around a particular methodology and cannot easily modify the process to fit other methodologies; and finally (c) Development managers are not intimately familiar with SDLC choices that are available to them and so continue to use the status quo methodology. Of the six companies surveyed, only one had different formal SDLCs for project development versus maintenance. That organization is using SCRUM for its project

development and KANBAN for its project maintenance. However, several had a formal methodology for development and an informal one for maintenance.

Table 1.
Criteria used in SDLC selection / modification

Company	SDLC selection / modification
Company A – Produces Software as a Service (Saas) Application	They use SCRUM for all projects both maintenance and new development. Critical maintenance updates use their own internal methodology.
Company B – Large Retail Store	The same SDLC was used for every project the SDLC had additional artifacts that were used depending upon the project type. Project size/cost and the departments involved guided which artifacts were used.
Company C – Large Transportation Company	The same SDLC was used for every project but risk, environment, number of people, number of departments involved, cost were all used to determine the amount of rigor applied to a given project.
Company D – Medium Membership Organization	They use an informal waterfall for all projects. No formal assessment is done and projects managers implement as they see fit.
Company E – Regulatory Compliance Application	They use SCRUM for all projects both maintenance and new development
Company F – Airline	They have a large portfolio of projects. Many are very small applications and some are enterprise wide ERP systems. They use a variety of methodologies. For larger projects that need multiple developers they use scrum. For small projects, they use a waterfall approach where the design work is done in advance by an analyst and then turned over to a developer when the project is scheduled. Maintenance follows the SDLC for the project or is waterfall for smaller items. Critical maintenance updates don't really have a methodology.

When asked if managers were familiar with the recent scholarly research regarding the selection of SDLCs, five out of six were not. Some of them were familiar with the research when they were studying for their undergraduate or graduate degrees; others did research as part of a previous work project; none were familiar with research carried out within the last 10 years. This is consistent with the development managers selecting a SDLC methodology for their team and then building repeatable processes around that SDLC rather than changing for different

projects. Managers were surveyed on what criteria they would consider if they were moving to a new company, creating an entirely new product, or were forced to reevaluate what SLDC they currently use. Their ranked responses are listed below. Interestingly, the release cycle was the only response indicated by all managers. This is probably because the managers being surveyed are not only responsible for selecting the SDLC but for building a repeatable process

1. Release Cycle: How often products or updates are released? (biweekly, quarterly, when ready, as soon as it can be fixed, etc.)
2. Product type: Is the product a website where they can have several updates, or a machine EPROM where it is costly to have more than one update?
3. Product/Project Risk: What is the risk to the business, users, others if the software doesn't work?
4. Team location: Are the users, developers all in the same location?
5. Team experience: Are the developers experienced in the business domain & software tools to be used?
6. Team Size: How many analysts, testers, developers are on the team?
7. Timeframe from development to release: How long from the time they start is the first deliverable due?

Using the CuQuP Framework from Yusof et al. (2011), quantitative scores were calculated for both standard project development and project maintenance for each of the companies interviewed. The CuQuP framework creates a total score for a number of SDLCs based upon user entered project complexity, project uncertainty, and quality criteria. I extended the framework to add Agile, Scrum, and Kanban to the methodologies supported. The top two or

three SDLCs with the highest total score should be evaluated by the project team to see which is the best fit for the company and project. Table 2 shows the recommended SDLCs in order for each of the companies. For most companies the same SDLCs appear in both the development and maintenance recommendations, but frequently in a different order.

Table 2.

Recommended SDLC

Company	Current SDLC	Recommendations for SDLCs	
		Development	Maintenance
Company A – Produces Software as a Service (Saas) Application	Scrum	Scrum, IE, RAD	Kanban, Scrum, Agile, IE
Company B – Large Retail Store	Proprietary	IE, RAD	Scrum, IE
Company C – Large Transportation Company	Proprietary	IE, Rad, Scrum	IE, Rad, Scrum
Company D – Medium Membership Organization	Waterfall	Scrum, IE	Scrum, IE, Kanban
Company E – Regulatory Compliance	Scrum	Scrum, IE	Scrum, IE, Kanban
Company F – Airline	Scrum	IE, Rad, Scrum	IE, Scrum, Rad

The recommended SDLCs and their order is slightly different for project development and project maintenance. Evaluating the CuQuP Framework and how its numeric scoring works, the difference in SDLC selection comes from the reduction in complexity and uncertainty for maintenance projects. This seems logical; maintenance projects have smaller scopes and durations. Changes that need to be made are usually clearer than building the initial system.

Section 4 - Conclusions

Using the surveys from a cross section of development companies and the scoring framework, it was interesting to see most companies were using a SDLC methodology that fit their organization well, even though the majority of them did not use SDLC evaluation

frameworks to select it. Selecting any SDLC is important to providing high quality software on time and on budget. Selecting the right one makes that possible in a repeatable way with a minimum of effort to the team.

Using a selection framework like CuQuP or others makes evaluation of various SDLCs easier. The traditional way of selecting a methodology based on familiarity is probably not the best approach. Selection frameworks take into account complexity, uncertainty, and quality requirements. They are also configurable to account for differences in projects or organizations. The framework provides the team several different SDLCs to consider. Frequently, this leads the team to learn more about different SDLCs and this knowledge can be fed back into the selection process and additional iterations of using the selection framework can be done to help make the best selection.

Through the research it was clear that project complexity and uncertainty are the main drivers of the selection of a SDLC. Projects that have low complexity and low uncertainty can be successfully completed with a lightweight methodology. Projects that have high complexity and high uncertainty need rigorous requirements definition, project management, and other work practices to be successful. Maintenance projects normally have lower complexity and uncertainty because the work that needs to be done is smaller in scope and is better understood. Because of the reduced complexity and uncertainty fewer rigorous work practices are necessary to deliver a quality software update. A comprehensive project management schedule or perhaps a daily standup may not be necessary to be successful.

All SDLCs, even an informal one, have some amount of overhead they add to the development process. Activities like creating specifications and test plans, resource planning, and risk analysis all take time and cost money to the organization. Therefore, selecting the

SDLC with the right amount of overhead will allow the organization to be successful delivering quality software in a repeatable fashion. Organizations should consider selecting a different SDLC for maintenance projects versus initial project creation.

The challenge for these organizations is implementation. A SDLC methodology does not exist in a vacuum. It is surrounded by people, processes, and tools. The implementation an SDLC into an organization is challenging. The analysts, developers, and managers need to be trained to use the methodology. Organizational processes need to be created to handle things like support tickets, rework, requirements gathering, and other activities that support internal and external customers. Software tools may need to be purchased to implement the methodology. These people, processes, and tools working in harmony around the SDLC are what make an organizational successful.

However, for an organization to select two SDLCs and implement them into their organization could be difficult. Although a lighter weight methodology might make sense, it has to be weighed against additional overhead incurred by having more processes surrounding the additional SDLC, additional training, and possibly additional capital outlays for more tools.

So the decision is not clear-cut. Organizations should thoughtfully evaluate the methodology they are using now and determine if it is meeting their needs. If the reduction in overhead by using a different maintenance SDLC outweighs the additional cost of having additional processes, then it probably makes sense to use the right tool for the job.

References

- April, A., & Abran, A. (2008). *Software maintenance management: Evaluation and continuous improvement (Practitioners)*. Hoboken, NJ: Wiley
- Avison, D., & Fitzgerald, G. (2006). *Information systems development: Methodologies, techniques and tools* (4th ed.). London, UK: McGraw-Hill.
- Burns, R., & Dennis, A. (1985). Selecting the appropriate application development methodology. *SIGMIS Database 17*, 19-23. doi:10.1145/1040694.1040696
- Cockburn, A. (2000). Selecting a project's methodology. *Software, IEEE 17*, 4. doi: 10.1109/52.854070 Retrieved from <http://www.eee.metu.edu.tr/~bilgen/Cockburn647.pdf>
- Cropley, D., Yi, Y., & Cook, S. (2003, October 28-30,). On identifying a methodology for land C2 architecture development. *Proceedings of the Land Warfare Conference, Adelaide, Australia*, pp. 401-409.
- Fitzgerald, B. (1998). An empirical investigation into adoption of systems development methodologies. *Information & Management 34*, 317-328. Retrieved from <http://ulir.ul.ie/bitstream/10344/103/3/08SE2037.pdf.txt>
- Gruner, S., Klopper, R., & Kourie, D. (2007). Assessment of a framework to compare software development methodologies. *Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information*, 56-65. doi:10.1145/1292491.1292498
- Little, T. (2005). Context-adaptive agility: Managing complexity and uncertainty. *IEEE Software 22*, 28-35. Retrieved from

<http://www.toddlittleweb.com/Papers/Context%20Adaptive%20Agility%20IEEE%20S3028.pdf>

Sarkar, S., Sindhgatta, R., & Pooloth, K. (2008). A collaborative platform for application knowledge management in software maintenance projects. *Proceedings of the 1st Bangalore Annual Compute Conference*. doi: 10.1145/1341771.1341774

Shine Technologies. (2003). *Agile methodologies - Survey results*. Retrieved from http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf

Standish Group. (1995). *The Standish Group Report – CHAOS*. Retrieved from <http://www.cs.nmt.edu/~cs328/reading/Standish.pdf>

Yusof, M., Shukur, Z., & Abdullah, A. (2011). CuQuP: A hybrid approach for selecting suitable information systems development methodology. *Information Technology Journal*. doi:10.3923/itj.2011 Retrieved from <http://docsdrive.com/pdfs/ansinet/itj/0000/23759-23759.pdf>

Appendix A. Questionnaire

Individual Background

- Name
- Title
- Educational background
- Years experience in development
- What kinds of SDLCs have you worked with?
 - How long?
 - How would you rate your expertise (1 being novice, 5 being expert)?
 - How familiar are you with the scholarly research on SDLCs (1 being novice, 5 being expert)?

Company Background

- Name
- Location
- Total number of employees
- Number of development locations / developers per location

Development Process

- How do you categorize projects?
- Is your standard development process different from your maintenance process?
 - How?
- What is your release cycle for standard development?
- What is your release cycle for maintenance?
- What SDLCs do you use?

- For how long?
- What did you do previously?
- Do you use a different SDLC for different projects?
- Do you use a different SDLC for maintenance?
- How do you decide which SDLC to use?
- Do you use any sort of framework to make the SDLC decision?
 - If yes, describe
- Who decides which SDLC to use?
- Does the project type impact the decision?
 - How?
- Does the project environment impact the decision?
 - How?
- Do you separate the business analysis process from the development process?
 - How?
 - Why?
- Are you development teams collocated?
 - If no, how do you facilitate team communication
- Do you use outsourced development teams?
 - Where?
 - How many?
 - How do you transfer requirements
 - How do you facilitate team communications
 - Do you select their SDLC or do they?

Appendix B Annotated Bibliography

April, A., & Abran, A. (2008). *Software maintenance management: Evaluation and continuous improvement (Practitioners)*. Hoboken, NJ: Wiley.

In this book, April and Abran discussed the frequently overlooked area of software maintenance. They discussed how companies do not have a defined software development lifecycle (SDLC) methodology for maintenance activities and showed how that leads to poor quality, project failures, and dissatisfied customers. They put forward the Software Maintenance Maturity Model (Sm^{mm}) based upon their research and interviews. This book is very important because it is one of the few that identify software maintenance is different from development and should be treated with different methodologies.

Avison, D., & Fitzgerald, G. (2006). *Information systems development: Methodologies, techniques and tools* (4th ed.). London, UK: McGraw-Hill,

One of the central topics of the book is reviewing different development methodologies. They also provided a comprehensive framework for comparing the methodologies based upon factors such as rules, scope, project size, and methodology outputs. Many authors have cited this book because of its treatment of methodologies and the framework to compare them.

Burns, R., & Dennis, A. (1985). Selecting the appropriate application development methodology. *SIGMIS Database* 17, 1, 19-23. doi: 10.1145/1040694.1040696

In 1985, R. N. Burns and A. R. Dennis introduced the idea of selecting an SDLC based upon the type of project. They introduced the key idea that the selection process should be made with a combination of project complexity and project uncertainty. Burns and Dennis are sentinel researchers who are cited by most of the other SDLC authors in this

bibliography. Their paper is important because it is one of the first papers to suggest the idea of selecting a project's SDLC based on complexity and risk.

Cockburn, A. (2000). Selecting a project's methodology. *Software, IEEE, 17*. doi:

10.1109/52.854070. Retrieved on 6/5/2011 from

<http://www.eee.metu.edu.tr/~bilgen/Cockburn647.pdf>

Alistair Cockburn has published several papers in software project methodology. In this paper he stated that "A larger group needs a larger methodology" (p.) and that the more critical the system, the more clearly defined functional decomposition that needs to go into its planning. He devised a ranking score based upon project risk, communication style, complexity, and project priorities to create a ranking score for a set of project methodologies being compared. This paper is useful because it shows another framework that can be applied to comparing project methodologies.

Cropley, D., Yi, Y., & Cook, S. (2003, October 28-30). On identifying a methodology for land C2 architecture development. *Proceedings of the Land Warfare Conference*, Adelaide, Australia, pp. 401-409.

Cropley, Yi, and Cook defined a process for creating a hybrid methodology for the creation of a Land C2 Architecture. They discovered that no single methodology was capable of creating the system they needed so they hypothesized a hybrid of existing methodologies that would address the particular needs of the project. This paper is one of the earlier works in using a hybrid methodology in execution of a project and is interesting to see how they use quantitative criteria to select methodology subcomponents.

Fitzgerald, B. (1998). An empirical investigation into adoption of systems development methodologies. *Information & Management*, 34, 317-328. Retrieved from <http://ulir.ul.ie/bitstream/10344/103/3/08SE2037.pdf.txt>

In this article, Fitzgerald reported on a survey he did through the University of College Cork in 1998. He was interested to find who was using a formal development methodology, to what extent were they using one, and the future trend. Fitzgerald did a formal survey of 776 individuals from various companies. Using a formal survey tool he split the companies into multiple cohorts and treated them statistically to find significant trends. His survey is important because it shows that in 1998, 60% of the companies he surveyed were not using a formal methodology. Now a little more than 10 years later many companies still do not have a formal methodology.

Klopper, R., Gruner, S., & Kourie, D. (2007). Assessment of a framework to compare software development methodologies. *Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, 56-65. doi: 10.1145/1292491.1292498

Klopper, Gruner, and Kourie created a framework to help make a decision about what software development methodology to use at a software engineering company. They used a variety of sources for candidate frameworks and settled on the one presented by Avison and Fitzgerald (2006) They took that framework and applied it to selecting a good software development lifecycle (SDLC) methodology for their organization. This is a useful article because it goes through the process of selecting a framework and then the actual application of the framework to select an SDLC methodology.

Little, T. (2005) Context-adaptive agility: managing complexity and uncertainty. *Software*

IEEE, 22, 3. Retrieved on from

[http://toddlittleweb.com/Papers/Context%20Adaptive%20Agility%20IEEE%20S3028.](http://toddlittleweb.com/Papers/Context%20Adaptive%20Agility%20IEEE%20S3028.pdf)

pdf

Little's paper is a good foundational paper that is referenced by many others. It calculates complexity using a 1, 3, 5, 7, 10 scale for mission criticality, team location, team capacity, domain knowledge gaps, and dependencies. Uncertainty is calculated using the same scoring range for market uncertainty, technical uncertainty, project duration, and dependencies. Using those factors different proscriptive development practices are selected to be used with the project. Little built upon the work of Cockburn and others to select specific practices instead of a whole SDLC methodology. This paper is helpful because it uses the idea of project classification and selecting from a pallet of development techniques to achieve the projects goals.

Sarkar, S., Sindhgatta, R., & Pooloth, K. (2008). A collaborative platform for application knowledge management in software maintenance projects. *Proceedings of the 1st Bangalore Annual Compute Conference (COMPUTE '08)*. doi:

10.1145/1341771.1341774

Sarkar, Sindhgatta, and Pooloth explained in this article the importance of managing and sharing domain specific knowledge to the success of a project. They developed a collaborative knowledge sharing tool called CollabDev and evaluated its use within projects. They found that by creating a way to code and store domain specific knowledge among multiple developers and in multiple languages allowed collocated teams to communicate more quickly and accurately. This is important because it is an example of

how collaboration tools can be used to enhance knowledge transfers and make project more efficient. Tools need to be considered as part of the environment when selecting a software development lifecycle (SDLC) methodology.

Shine Technologies. (2003). *Agile methodologies - Survey results*. Retrieved from

http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf

In this report of survey results, Shine Technologies, described their experience and their customers' experience with agile development methodologies. Shine Technologies is a Software Development company based in Victoria, Australia that ran a web based survey of 131 respondents. They designed a survey tool to ask 10 questions to gauge the experience, interest, and success or failure of the individual with agile development methodologies. This article is useful because it demonstrates that the selection of an appropriate methodology can significantly improve product quality, reduce cost, and increase business satisfaction.

Standish Group. (1995) *The Standish Group Report – CHAOS*, Retrieved from

<http://www.cs.nmt.edu/~cs328/reading/Standish.pdf>

The CHAOS report is a survey of information technology companies and the number of software projects that fail either by not being delivered at all or by being delivered over time and/or over budget. The Standish group is an IT consulting group formed to collect case information on it project success, failures, and ways to improve the former.

Standish used a detailed methodology to combine research surveys and qualitative personal interviews of a cross-section of development companies. They surveyed project success factors and project challenge factors and combined those into a success criteria matrix that included user involvement, executive management support, clear statement of

requirements, etc. This is an interesting paper because it lists several factors that determine failure, and therefore, risk that are part of a project and how they can be mitigated.

Yusof, M., Shukur, Z., & Abdullah, A. (2011). CuQuP: A hybrid approach for selecting suitable information systems development methodology. *Information Technology Journal*.

Retrieved from <http://docsdrive.com/pdfs/ansinet/itj/0000/23759-23759.pdf>

In this paper, Yusof, Shukur, and Abdullah introduced a hybrid approach of selecting a SDLC methodology based upon complexity level, uncertainty, quality criterion and methodology. They reviewed the landmark work of Burns and Dennis (1985), Cockburn (2000), and Cropley et al. (2003) and finally proposed their new approach, CuQuP, which is a distillation of the Avison and Fitzgerald (2006) framework to use complexity, risk, quality requirements, to compare different methodologies using a quantitative scoring method. This is an important paper because it introduces the idea of using quantitative methodologies in the selection process. I use their framework in comparing the different methodologies for the companies surveyed.