Capstone Title:

The Systems Development Life Cycle: Phases and Methodologies

Student Name: Erik Paul Harrelson

Faculty Supervisor: Dr. Charles Downing

Faculty Approval Signature

Department of Operations Management and Information Systems

Date of Approval 4/23/10
In order to obtain a permit for OMIS 495, please turn in your proposal (using the following guidelines) to the OMIS office at least a week before the deadline for mail registration. This sheet should be affixed to the top of your proposal.

**Proposal Guidelines**

1. Explain your problem statement. What is the purpose of your study?

2. Give a rational for the project you are proposing. For example: Why have you chosen this topic? Why do you feel it is important? What do you think your findings will contribute to the field/industry?

3. Describe the methodology you will use to research your problem statement. What data or information do you plan to use? How do you plan to collect it? How will you analyze it?

4. Obtain the agreement of the professor who will be working with you on the project and have the professor complete the section below.

**Professor Agreement**

I agree to sponsor this student's independent study, monitor his/her progress throughout the term, and submit a final grade in his/her behalf.

Signature of Sponsoring Professor

Date

Signature of OMIS Advisor

Date
REQUEST FOR UNIVERSITY HONORS INDEPENDENT STUDY LEADING TO THE COMPLETION OF THE HONORS CAPSTONE PROJECT

COVER SHEET

Erik Harrelson
Student Name

E: harrelsg@nlu.edu
E-Mail Address

OM 315 495
Department and Course Number

Semester of Registration

May 2010
Graduation Date

Local Address

DeKalb 60115
City, Zip

Student Phone Number

Date of Request
Signature Page

Student Signature

Student Z-ID#

Request Approved:

Charles E. Downing

Printed Name of Faculty Capstone Advisor

11/11/09

Date of Acceptance by Faculty Capstone Advisor

Chair

Printed Name of Department Chairperson

12/21/09

Date of Acceptance by Chairperson

University Honors Program Director

Signature

Date of Acceptance by Director
AUTHOR: Erik Harrelson

THESIS TITLE: The Systems Development Life Cycle: Phases and Methodologies

ADVISOR: Dr. Charles Downing

ADVISOR'S DEPARTMENT: Operations Management and Information Systems

DISCIPLINE: Information Systems

YEAR: 2010

PAGE LENGTH: 11

BIBLIOGRAPHY: Yes

ILLUSTRATED: No

PUBLISHED (YES OR NO): No

LIST PUBLICATION: N/A

COPIES AVAILABLE: On CD-Rom

ABSTRACT (100-200 WORDS): The essay focuses on introducing the Systems Development Life Cycle to those who are unfamiliar with it. It is meant for both Information Technology Professionals as well as individuals with little experience in Information Technology. The essay analyzes the four central methodologies from which all system design plans stem. In addition, the essay also reviews different phases contained in different Systems Development Life Cycles and explains how there is no one size fits all solution for every project.
The focus of the report is to create an overview for steps to take toward successful system development. It will be useful for students, professors, and firms due to the flexibility of the Systems Development Life Cycle. An important consideration is that every company has a different version of the Systems Development Life Cycle, yet they all refer to extremely similar concepts. This report will address many of the commonly used SDLC concepts, while creating a logical unification of ideas. The SDLC is important because without it creating systems would be much more difficult, if not impossible.

Establishing an order of operations when creating software is important because it reduces scope creep and makes estimating the amount of time to allocate to each phase easier. Scope creep is when features are added to software that were not originally intended to be included. Reducing scope creep and making accurate estimates is essential to software development because programmers can only do a limited amount of programming per day. Given infinite time a perfect program could be created, but in today’s competitive world time is extremely limited. This means programmers need to remain focused and be given a clear set of instructions on what type of software they are building, and what features must be included. It is always possible to eliminate features if time is running short, but the important thing to focus on is not starting on features that will not be finished, or will not be well made. Following a plan (such as the SDLC) is an excellent way to avoid the previously mentioned pitfalls.

The study will be part qualitative and part quantitative. It is sometimes difficult to quantify data relating to the SDLC, yet it is important to do so in order to prove that the SDLC creates positive results. Quantitative methods will be used when possible, and qualitative methods will be used the rest of the time. Qualitative methods are much simpler in this case because processes are being analyzed. The
The focus of the report is analyzing standard SDLC methods which improve the quality and creation time of systems that are being developed. I will be using internet research due to the wide variety of information available. The internet is also a fitting source due to the nature of the report: looking at methodologies for successful system and software creation. I will analyze actual SDLCs that have been created by actual firms and organizations. Writings by information technology professionals will also be analyzed in order to discover ideas which may be popular and sensible, yet have not become mainstream within most SDLCs. The goal is to create a well rounded report which will serve a variety of users; in order to create something useful for so many different users a variety of sources must be consulted.

I have no need for external funding. I will use the internet as my source in order to conduct research, and will create a report based upon my findings. Milestones include completing the proposal before finals in December, 2009. Work will begin on the report the first week of the Spring semester. Within two months of the start of the semester the report will be written, followed by proofreading and editing. The report will be delivered to the Honors Office Revision Committee by the end of the first week in April, 2010. Corrections will be made in accordance with the committee’s recommendations, and the final report will be submitted soon thereafter.

OMIS 462 provided me the idea for this study. The Systems Development Life Cycle was a central concept in that course which focused on systems development. Dr. Charles Downing did an excellent job teaching the class; therefore I chose him to be my advisor for the report. He is extremely experienced with the SDLC and is a very knowledgeable resource to aid in my research. Because I am a Management Information Systems major this study will greatly improve my understanding of how systems are built. Understanding all aspects of the system creation process will enable me to adapt to real world business practices easily, and therefore I will be more valuable as a worker to my future
employers. My internship experience with Hub Group over the summer revealed how different school and the business world are. While teachers at Northern Illinois University do an excellent job of speaking about the real world there is no substitute for actual experience. This study will enable me to adapt to the real world and become productive faster than otherwise possible.
Works Cited


<www.hhs.gov/ocio/eplc/eplc_framework_v1point2.pdf>.


<http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci755068,00.html>.
TABLE OF CONTENTS

INTRODUCTION ............................................................................................................................................. 1
   Existing SDLC Methodologies and Processes ....................................................................................... 1
   Scope of the Study ............................................................................................................................... 2
   Sources and Methods ......................................................................................................................... 2

SDLC Methodologies ............................................................................................................................... 3
   Waterfall Model .................................................................................................................................. 3
   Spiral Model ....................................................................................................................................... 3
   Rapid Prototyping ............................................................................................................................. 4
   Build and Fix ...................................................................................................................................... 5

SDLC Phase Analysis ................................................................................................................................. 5
   Planning ............................................................................................................................................ 5
   Analysis .............................................................................................................................................. 6
   Design .............................................................................................................................................. 7
   Implementation ................................................................................................................................. 8

CONCLUSION ............................................................................................................................................... 9

REFERENCES ............................................................................................................................................ 11

Appendix A ............................................................................................................................................... 13

Appendix B .............................................................................................................................................. 14

LIST OF FIGURES

Figure

1 Maryland's 10 Step Phase Layout ....................................................................................................... 2
2 SDLC Phases Visualization .................................................................................................................. 14
The Systems Development Life Cycle: Phases and Methodologies

INTRODUCTION

According to MKS (an application lifecycle management firm) the Systems Development Lifecycle is: "The entire process of formal, logical steps taken to develop a software product". This is a generic definition for a very complicated series of processes, yet it is a great place to begin discussion of the Systems Development Life Cycle. The Systems Development Lifecycle (abbreviated as SDLC) is an extremely flexible set of methodologies which thousands of firms utilize in order to successfully complete software, website, and other technology based development projects. What the SDLC does is give the team guidelines including how much utilizing previously coded work is allowed, how much project sponsor/stakeholder feedback there will be, and what order different modules within the project should be coded.

The SDLC is very interesting because there are nearly infinite variations of it. Some companies may rely entirely on older development methodologies such as Waterfall, while others may use newer techniques such as Rapid Prototyping. The important thing to consider is that there is no right or wrong answer for every company. Waterfall may be the best solution for one company, while another company may be better served by Rapid Prototyping. Even within a single company the best methodology may vary by project. This means that developers, project managers, and everyone else involved with development of an application must be knowledgeable about the SDLC, or open to the idea of the SDLC. The SDLC methodologies are well understood, and therefore it is not difficult to discover what other companies are doing, and what has or hasn’t worked for them.

Existing SDLC Methodologies and Phases

While there are a plethora of countless sub-methodologies, there are four core methodologies including: the waterfall model, the spiral model, rapid prototyping, and build and fix (according to GIAC, the Global Information Assurance Certification website). These four methodologies allow a variety of strategy within firms, and are extremely adaptable to needs of nearly any project. While adapting may cause these strategies to look more like a sub-methodology, it is important to realize that these four strategies make up the core of any other sub-methodology.

In addition to the different methodologies there are also different project phases. Different groups believe there can be anywhere from four to more than ten phases. Despite such a variance in quantity of phases, most groups have extremely similar phase layouts. The four core phases generally include planning, analysis, design, and implementation. These four phases are very broad, and can have dozens of sub topics, hence why many companies divide them further. One example of a much larger phase layout is taken from the Department of Information Technology for Maryland.
Maryland’s SDLC demonstrates what was referred to previously: a large diagram with 10 steps that still primarily revolves around the four key phases: planning, analysis, design, and implementation. While the diagram may look complicated initially, it is very similar to many other common SDLCs.

**Scope of the Study**

This study seeks to analyze what methodologies and phases existing companies and respected theorists believe in and utilize. The study is intended to be informative, rather than conclusive. As stated previously there is no best SDLC method. Optimal strategy depends on nearly countless factors, the most important being the project being undertaken and the humans working on the project. The SDLC is extremely useful for those who understand its phases: countless hours and dollars can be saved by choosing the correct methodology and utilizing it.

**Sources and Methods**

Due to the SDLC being a technology concept, the internet was used as my primary source of research. Both respected individuals, as well as actual companies using the SDLC were considered. There is valuable information both in theory and practice, and theory could point out why actual SDLCs may have succeeded, failed, or not been optimally executed.
SDLC Methodologies

Waterfall Model

The waterfall model is considered to be the oldest and perhaps most well understood of the SDLC methodologies. Waterfall operates by having one step entirely completed, and then moving onto the next. It is called waterfall because one can imagine a development team working on a project, finishing one stage, and then going down the waterfall. After plunging down, the team no longer has access to the previous elevation, and therefore can only work with what they currently have. While this methodology may seem foolproof, it has many flaws that become evident once actual development environments are taken into consideration. A study from ComputerWorld cites the primary problems extremely well: "Another problem is that the waterfall model assumes that the only role for users is in specifying requirements, and that all requirements can be specified in advance. Unfortunately, requirements grow and change throughout the process and beyond, calling for considerable feedback and iterative consultation". This quote cites how critical user feedback is. A large majority of projects are carried out not for the development team, but for end users or project sponsors. Therefore the sponsor's satisfaction should be the number one priority, a very difficult goal if the sponsor is not able to have any input into the system's design after the initial requirements are laid out.

While the sponsor will have little, if any, input into the system once the initial plans are developed, they are involved initially since they are the one requesting the system. The benefit of this is that the development team optimally will have an extremely firm understanding of what kind of system they are building, and what the key features need to be. The downside is that the team has no interaction with the sponsor once the project has begun. This means that the team has been guided, but is still somewhat guessing at what the sponsor wants, depending upon how specific the requirements were and how well they were communicated to the development team. Later methodologies have attempted to overcome these shortcomings and have succeeded while also bringing new problems along.

Spiral Model

According to ComputerWorld the spiral model “Emphasizes the need to go back and reiterate earlier stages a number of times as the project progresses”. In other words, spiral aims to fix the problem that waterfall had with completely sealing off earlier portions of the project and exclusively working toward completing steps further down the waterfall. The primary benefit of using the spiral model is the addition of user feedback. The ability to return to previous work enables basic prototyping where the development team can show sponsors what they have created, and then make changes based upon the sponsor's feelings. One important difference between spiral and other prototyping methods is that spiral typically uses the same software during prototyping and making the final product. Once the
sponsoring the team can then begin working toward the final system that the sponsor will actually use within their company or organization based upon the sponsor's favored prototype.

While the spiral model adds tremendous flexibility there is a cost: projects can easily become sidetracked. Development teams may be left scrambling trying to meet updated system requirements provided by the sponsor. In addition, the sponsor may want more additional functions that are not critical to the system. This may put the team behind, and lead to core portions of the system not being completed or being completed to lower standards or less functionality than initially planned. The formal term for additional requirements being added after the initial conception of a project is scope creep. An individual or the entire team must ensure that the sponsor's needs are understood while not overburdening the developers. This is a very difficult balance to find, but if performed properly the sponsor will receive the useful system they want on time.

**Rapid Prototyping**

Rapid prototyping is somewhat similar to the spiral model in that both are highly iterative processes (many versions of the same system) that both utilize prototyping. Rapid prototyping differs, however, in two primary ways: the first difference is that rapid prototyping focuses primarily on creating prototypes extremely quickly. Teams that use this method show sponsors many different potential versions of the system they requested, and allow the sponsors to pick their favorite. This gives the sponsors an abundant number of choices, although each individual system may be less developed than if fewer prototypes were produced. This overall choice is beneficial to sponsors since they may have had an idea of what they were looking for, but not have been exactly sure. The disadvantage is that the team worked on creating many prototypes, only one of which will actually be used. The rest of the prototypes are minimally useful and serve very few purposes, although one use could be as a proposal to other clients.

The second key difference between rapid prototyping and other methodologies such as the spiral model is that rapid prototyping tends to use different development packages between the prototypes and the final product. This makes sense for rapid prototyping because some tools will allow the team to develop simple systems for prototypes quickly; while other tools will allow more time to be spent creating detailed final products. The focus when using different development tools is that they need to make sense for the team, just like any good SDLC will. It makes little sense, for example, to require a team to learn a new programming language just to create prototypes. On the other hand, if the team can effectively use multiple tools which are optimized toward different development stages the team will be able to create prototypes and finished systems extremely efficiently at a high level of quality.
Build and Fix

Build and fix is much different from the three previous methodologies explained previously. Build and fix works in reverse of how most other methodologies do; namely in that systems are created without nearly as much guidance as waterfall or the other two methods, and then the system is constantly updated (fixed) until the sponsor is happy. While many individuals outside the information technology world may think this is how development is typically done, this method is tremendously risky. Having less guidance from the sponsor is bad, because they should be the largest driver of the system being built. In this case the team has some guidance, but still has a reasonable voice into what the system will contain. If the team is highly experienced they may have a good idea of what the sponsor is looking for, but on inexperienced teams this development method can be catastrophic. It is important to keep in mind that every minute the team spends developing a feature the sponsor doesn’t want is costing the team time and money. These quantities are very limited, and directly take away from time that can be spent producing what the sponsor does want.

As mentioned previously, build and fix is an extremely risky method of development. Because there is so little planning with the sponsor the team must be extremely sensitive toward scope creep as well as monitoring time. Without a firm goal set in place it is very easy to fall behind schedule, and be unable to catch up. Of all the methods build and fix may be the most vulnerable to falling behind since it is extremely open-ended. The benefit of being so open ended is that if the project goes well there can be tremendous benefits. The first is that since there were no prototypes an incredible amount of time can be saved if the development team creates a system the sponsor likes. The second benefit is that because the team did not spend as much time as other methodologies in the project requirements gathering phase there can be both great cost and time savings.

In the end, it comes down to the type of system and members of the team creating the system to determine whether or not build and fix is a good fit. If the team is confident in their skills as well as the project definition given by the sponsor, build and fix can be a good choice. If the team is not able to create a system the sponsor likes, however, the team can quickly find itself far behind schedule with little to show. This method is a very high risk, high reward gamble which sometimes will pay off and sometimes will not.

SDLC Phase Analysis

Planning

Planning is always the first phase in the SDLC. Planning can include countless steps, but it typically begins with an analysis of the company and general ideas of how it can be improved via technology. This company analysis can view the entire company, a division of the company, or even a small portion of a division. Regardless of size, improvements can be made to improve efficiency, cut costs, or make tasks
easier to perform for workers. Once there is an idea of what should be improved, how to improve it must be discovered next. In order to solve this problem a project manager is appointed. This person is the head individual who oversees progress on the project, tries to prevent scope creep, and plays a large part in guiding the overall direction of the entire project. The project manager is usually the driving force behind the project, and typically has immense power regarding the scope of the project, the team selected to complete the project, and the timelines regarding project completion. Due to the project manager’s immense power there is a great chance of being fired if the project is a failure, or being promoted if the project is successful. This pressure is typically a good thing, because it will drive project managers to create the absolute best project, deliver it on time, and ensure satisfaction to those who use it.

Once a project manager is appointed there are a multitude of steps to be taken next including: selection of the project team, deciding what steps will be included in the project, and figuring out very tentative dates for deliverables and other milestones. Deciding what steps will be included is related closely to the rest of the SDLC, and is very detailed. Some projects may need tremendous amounts of planning before any work can be started on the project itself, whereas other projects may have individuals on the staff that can begin programming the same day the project idea is realized. Regardless of the overall quantity of planning, how the new system will fit in needs to be analyzed, and there must be maintenance and updates once the system is live.

Creating time estimates is an essential part of the planning process. While it is extremely difficult to assign an amount of time to a project, doing so is highly valued in the business world. Bosses need to know how long projects are going to take in order to approve or deny them, as well as plan for future projects by looking at worker availability. The nice part of time estimates is that they are just estimates. The reality of estimates is that they tend to be underestimated, yet bosses obviously like to see expectations exceeded rather than fall short. Bosses also want projects to be completed as quickly as possible while maintaining quality. This clearly leads to conflict, but is a struggle that skilled teams overcome by keeping open communication, allowing revision of deliverable dates (based upon how long tasks are actually taking), and having an understanding between departments such as management and the developers creating the system.

**Analysis**

Analysis is the beginning of creating system related deliverables for the project. This is where everything done in the planning phase is reevaluated in context of the project, as well as where many diagrams and forms are created in relation to the project and proposed system including entity relationship diagrams, dataflow diagrams, and many other documents which analyze the current processes of the company/system.

Entity relationship diagrams are typically used to serve as a visual depiction of a database. Similar to the project scope these can be a single page with three tables, or can be multiple pages with thousands of tables. One of the best aspects of entity relationship diagrams is that it is simple to see the connections between different entities within databases. These connections serve three purposes: the first is that
the diagram serves as an easily viewable guide that can be looked at without requiring access to the actual databases running within the system or that will be created to work alongside the future system. The second purpose entity relationship diagrams serve, is to allow easy analysis of the database logic while building the system. Determining whether relationships between entities are correct is much easier when there is an actual picture diagram rather than Structured Query Language and database tables. The final primary use of entity relationship diagrams is to not only show a visual representation of the databases, but to also show relationships between individual tables in the form of cardinality and modality. Cardinality represents the maximum number of times an instance in one entity can be associated with instances in the related entity, whereas modality represents the minimum number of times an instance in one entity is associated with instances in the related entity. The importance of cardinality and modality is in how they may reveal that additional tables are needed. If there are many to many relationships there may be a need to have more tables, and doing so will optimize the database when queries are run.

Dataflow diagrams show how data travels through a system. There are a large variety of dataflow diagram styles, but typically data stores, data flows, processes, and external entities are the objects that data flow diagrams revolve around. Each diagram focuses on a different level of the system, such as the level 0 which focuses on the entire system at a very high level. Increasing levels then continue to get more specific until the diagram cannot be more specific (or doing so would serve no purpose). The value of these diagrams is that they can give a development team a great place to start with while looking at what the project requires, and these diagrams also enable the project to be understood much more easily than on a verbal basis alone. This visualization enables presentations to be made to supervisors, and serve as concrete evidence that there is a firm understanding of what a system requires. An additional benefit of creating dataflow diagrams is that inconsistencies will become evident rapidly; if an important module with many dependant modules is drawing information from an outside source it will be clear to anyone viewing the diagram. This may save the company time and money later on if they understand the importance of each module, and how many dependencies there are upon each part of the system.

Design

The design phase utilizes all the planning and analysis from previous phases, and signifies the beginning of actual creation of the system. Based upon previous decisions, the team creates a system which will (hopefully) become the actual system used by the organization. Perhaps the biggest decision is which program will be used for development of the system. Decisions made in previous phases should guide which development tool will be used, alongside what existing skills the development team has, but development software selection is still one of the most important decisions that can decide whether a project is a success or failure. One large complicating factor to this problem is that while there may be an obvious choice for a development platform which should be used, outside factors such as manager’s preferences, or company standards can get in the way. An example is if Visual Studio might be the best program for a system to be developed on, but the company does not own it and therefore does not want to pay for the program. This is a direct confrontation between what’s best for the project, and
what is best for the company’s bottom line, and therefore it may be difficult for the company to make the choice to purchase Visual Studio.

In addition to deciding which tool to use for development, the team faces the important issue of how the new and old systems will coexist (if at all). There are numerous plans for integrating a new system, and these will be further discussed in the implementation section. While implementing the new system is a huge issue, there are also many other issues which need to be thought about before the implementation phase. Some potential issues include what the development team will do if they have trouble integrating the new system with older systems, how to transition existing employees from using the old system to using the new system, creating documentation in order to show employees and other users how the system works, demonstrating how the new system is an improvement from the old system, and what it means for employees.

Integration with previously existing systems is a huge facet of system design. While the new system may be an integral part of the organization, it is still only one piece. Therefore it is important to ensure the new system will be compatible with other modules it may be required to work with. A benefit of this is that it will make the transition to the new system easier for existing employees, since previously used information and formatting should be very familiar, therefore reducing employees’ resistance to change. Humans are naturally resistant to change, especially if they are required to change from a system they might use eight hours a day. There are a variety of strategies to create a positive reception toward the change, but the key is to tell the employees why the change was made, how the new system will make their life easier, and how it is beneficial to the company overall. Humans are intelligent, and anything that makes their lives easier will be greatly appreciated; especially if it is also helping the company (and therefore improving job security). An additional tool to assist in employee’s transition to the new system is documentation. Well-written manuals can enable users to easily adapt to the new system, although poorly written manuals can have the opposite effect. A well-written manual can tell users about the basic functions they need to know, as well as higher level functions which some users will need to know, but others may not ever need. This segregation of knowledge needed by different users is a large determinant in the quality of a manual; if users of varying amount of features can all find what they need to know the manual is probably excellent, whereas if users are unsure how to work with the system regardless of level the manual is probably poor.

Implementation

Implementation is the final phase of the Systems Development Life Cycle and includes putting the developed system into production. Implementation also includes replacing/slowly phasing out the previous system, and eventually using the new system exclusively. Many consider testing to be a large portion of implementation because testing in a closed environment with dedicated testers is one thing, but going live and experiencing actual use by regular employees can lead to many previously unforeseen issues relating to both user confusion as well as number of users. In addition to testing and implementing the system, this is also the step where the documentation must be made available to
potential users of the system, data is likely to be copied or transferred from the old system, and employees must begin transitioning from the old system to the new one. Data transfer is a tricky issue, because duplicating data takes time, memory, and effort, yet an organization cannot afford to lose data while converting from one system to another. Typically firms will start with transferring all data while keeping a backup, but when new system has been proven stable less caution may be taken. Once the new system is fully in place transferring data will obviously become a non issue, but until the old system is entirely phased out the organization must have a firm plan for managing how data will be transferred or communicated between the two systems.

In addition to the initial struggles that the team will face in implementing the system, there are three final hurdles: maintaining the system, providing updates, and optionally discontinuing use of the old system (based upon whether it was previously discontinued or not). Maintaining the system largely depends on the system itself: some may require little to no maintenance, while others may require hourly checkups to ensure the system is working properly and efficiently. Updates to a system can serve as a great compliment to maintenance. If a system has a consistent issue that repeatedly needs to be fixed, updates can serve as a permanent cure to the issue. Updates can also perform a variety of functions such as improving the user experience, providing improved security, and making the system work more efficiently overall. Typically there is a combination of updates and maintenance, although individual systems may call for extreme amounts of maintenance, or zero maintenance and many updates.

Finally, the old system must be discontinued if use has not already been halted. As mentioned previously data transfer between systems varies largely by system, but when a new system is developed there is typically a 1 year or shorter time frame until the old system will be completely discontinued. Depending upon the system this time period can even be immediate, but typically organizations like to test the new system to make sure it is working properly, as well as ensure employees are comfortable with using the new system. The entire process is largely a matter of the system in question as well as the organization it will be effecting, so while generalizations can be made there will always be exceptions.

**Conclusion**

The primary goal of this report has been to analyze the variety of Systems Development Life Cycle methodologies as well as the different phases that are typically used. As discussed there are four main methodologies from which the other sub methodologies stem: Waterfall, Spiral, Rapid Prototyping, and Build and Fix. Each of these has unique differences, and each is best suited for different project types than the others (see Appendix A). The most important considerations when attempting to pick the optimal methodology is what the project requires and the resources the team has to construct the system. There is no “best” strategy every time because of the variety of projects that are created, as well as the variety between the different methodologies. Some methodologies such as Waterfall are best for projects in which the client has laid out extremely specific, clear details about the system being developed. Others such as Rapid Prototyping are much better suited for clients who are unsure of what they want. Another important thing for project champions to remember is that there are many strategies other than the ones presented here, as well as many that may not have been discovered yet.
The key to finding success is to ensure that the methodology selected fits the project well and will lead to the optimal business solution.

The SDLC phases are very similar to the methodologies because the phases are similar in many projects, yet there is no perfect model. Phases are an efficient method of attempting to give structure to a project in order to allow the team to understand how far along the project is. While four to ten phases are used, detailed phase models will contain many more sub steps in four phase models that tend to include many of the phases found within ten phase models. The net result is that many models are extremely similar despite initially appearing very different from one another. These differences are a positive force, since the phases differ between projects in order to allow project sponsors and developers to better understand how far they are on the processes required to create the system being developed. It is much easier to apply phases to a project than it is to strictly try and break up a project into exact phases.

The end result of understanding the SDLC is to allow creation of the best system possible given constricted resources. Businesses today put a tremendous value on time, sometimes to the point of believing that a system without all key features is acceptable if it is delivered on time. The SDLC aims to allow firms to have a quality system that is delivered on time, but it does have limitations. Leaders must understand the SDLC and have knowledge of both the basic methodologies and phases. Once the SDLC is understood system development will be much more efficient, and development teams and project managers will better understand the overall flow of the project. This project understanding will allow leaders to make the optimal decisions in order to create the best system possible using the least resources and least time.
References


<http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci755068,00.html>.
Appendix A

Figure 2

This is a visual depiction of the SDLC methodologies discussed in this study. It visibly shows the similarity between the spiral model and rapid prototyping, and the differences between all four methodologies. As discussed in the study, rapid prototyping and the spiral model both allow high amounts of returning to previous work while demanding low to medium amounts of client knowledge. Build and fix demands very low client knowledge while allowing low to medium amounts of returning to previous work, and waterfall requires high amounts of knowledge about the system the client wants but has minimal, if any, returning to previous work. This chart allows for easy explanation of why a methodology might be optimal for a project if it is possible to identify how much knowledge the client has, and how much returning to previous work the project manager and developers choose to do.
Appendix B

OMIS 259 System Project

The other portion of my upper division honors capstone was creation of a system which will be used for the upcoming OMIS 259 class at Northern Illinois University. In creating the system I utilized the SDLC to create the best product for my client Dr. Charles Downing. Due to this being a solo project I was the project manager, developer, and tester all in one. Dr. Downing initially had a medium amount of knowledge about the system he wanted created. He told me the basic functions desired such as writing reviews for students. I then took my knowledge of Visual Basic and began developing the system. In order to allow users to create reviews I integrated two database tables into the system: one which stores the users who have registered on the site, and one that stores the reviews written by users.

The SDLC methodology I used is a hybrid of Build and Fix and the Spiral Model. Dr. Downing had a good idea of what he wanted, although he didn’t give me specifics on everything, and there was a low to medium amount of returning to previous work. I worked on one or two web forms at a time, and made incremental progress reports to Dr. Downing. He then suggested any fixes that needed to be made, and made a few specific requests for additional features not originally planned. Fortunately his requests were modest and I would not consider them to be scope creep which negatively affected the project.

I am very happy with the methodologies I selected. Waterfall would have been a poor choice for this project because Dr. Downing did not give me enough concrete information that would make me perfectly confident that I was creating everything he wanted in the system. Rapid Prototyping was not a viable choice due to my development experience. I am only familiar with Visual Studio for website development, and learning another website development program is far beyond the scope of this project. Combining spiral and build and fix was excellent because it allowed me to do many things on my own (spiral) and then return to make changes once I had shown Dr. Downing my work (build and fix).

The amount I learned from building the system is incredible. I increased my knowledge in website design, working with databases, learning how the SDLC applies to actual projects, and working with a client in order to create exactly what he desired. The SDLC is certainly applicable to real world projects, and is something I will definitely use throughout my professional career. Everyone who does anything related to information technology should be familiar with it, as should managers who work together with information technology professionals. Knowing about the SDLC will save companies great amounts of time, money, and energy, and that is a perfect reason to be educated about it.
References


<www.hhs.gov/ocio/eplc/eplc_framework_v1point2.pdf>.


<http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci755068,00.html>.