Mainstream Information System Development Methods. Is One the Best or Does Each Serve A Unique Function?

ISAS 630 9041

Jeffrey Semon

This paper examines three mainstream information system development methods. The Systems Development Life Cycle (SDLC), the Agile development method, and the object-oriented method of design. The question applied to this research is if one of the models stands out as truly the best or does each model have its own advantages for certain situations.

The Systems Development Life Cycle (SDLC) is a phased approach to developing an information system in a cycle. Each phase integrates with one another and activities can occur simultaneously and may be repeated. There are seven phases that the SDLC can be broken down into. The first thing that should be considered is incorporating Human-Computer interaction considerations.

Human-Computer interaction (HCI) focuses on the communication and interaction between humans and computers. In the approach to HCI analysts focus on the people rather than work that is needed to be done. Kendall and Kendall tell us that the approach involves looking at factors such as ergonomics, the human thinking process and how users behave in their tasks with the system (2008 pp. 10). This is a move away from focusing on the businesses needs and more on the human needs. HCI focuses on what the human needs are and sifts out

the business or cultural needs to provide needs based perspective from the user. This ultimately improves the quality of the systems and work life (Kendall and Kendall, 2008, pp. 11). HCI focuses on addressing the concerns that users state about their use of the IT. According to Kendall and Kendall these concerns include,

> suspicion that the systems analysts misunderstand the work being done, the tasks involved, and how they can best be supported; a feeling of helplessness or lack of control when working with the system; intentional breaches of privacy; trouble navigating through the system screens and menus; and a general mismatch between the system designed and the way the users themselves think of their work process (2008, pp.11).

HCI helps remove system design errors that cause user neglect or systems failure soon after implementation. As mentioned earlier the HCI phases are integrated into one another and HCI is something that can be used in every phase of the SDLC.

The first phase of the SDLC is indentifying problems, opportunities, and objectives. This is the most critical phase of the SDLC because in this phase the analyst can make sure that the right problem is being addressed so that later in the project the team doesn't realize they are working on the wrong problem (Kendall, Kendall, 2008, pp. 11). In this phase the analyst will take an overview of the business and incorporate the business member's feedback to find the problems. Opportunities arise when there is a situation that computerized information systems can improve (Kendall, Kendall, 2008, pp. 11). When identifying objectives, the analyst will contemplate what the business is trying to accomplish. Once the analyst has done this the

analyst will be able to determine what problems and opportunities will be necessary to pursue in moving forward in the development of the information system. The first phase of this project will involve users, analysts, and program managers. According to Kendall and Kendall activities in this phase consist of "interviewing user management, summarizing the knowledge obtained, estimating the scope of the project, and documenting results" (2008, pp.11). The results are a feasibility report which defines the problems and summarizes objectives. This will be the basis for what management uses in their decision on how to proceed with the project.

The second phase in the SDLC is determining human information requirements. The analyst will be determining how the users interact with the information system in their work environment by deploying a variety of tools. According to Kendall and Kendall these tools include, "interviewing, sampling and investigating hard data, and questionnaires along with unobtrusive methods, such as observing decision makers' behavior and their office environments, and all-encompassing methods, such as prototyping" (2008 pp. 12). These methods will help the analyst develop questions concerning user physical strengths and limitations that directly address characteristics of the system concerning how audible, legible and safe the system is, how to make the system easier to use learn and remember, what can make the system pleasant and fun and how can the system be tailored to individuals tastes to make them more productive (Kendall, Kendall, 2008, pp. 12). This is the user needs phase of the SDLC. This is the phase where the analyst will be trying to determine what user needs are. The analyst will be determining how to design the system to make it more useful to users, design the system to make it more productive at accomplishing tasks, what are the new tasks available as a result of the new design, what new capabilities the system has over the old

system, and how can the system be rewarding to use (Kendall, Kendall, 2008, pp. 12). In this phase the analyst and users will be working together to enlighten the system analyst of current system functions. According to Kendall and Kendall this includes questions such as,

> The who (people who are involved), what (the business activity), where (the environment in which the work takes place), when (the timing), and how (how the procedures are performed) of the business under study. The analyst must then ask why the business uses the current system (2008 pp. 12).

Since there may be current efficiencies with the current system it is good to know why the business is currently using the system for consideration in the new design. At the end of this phase the analyst should understand the uses interactions with the computer and how it can be improved upon and how the business operates at this point.

The next phase of the SDLC is the analysis of system needs. The analyst will utilize special tools and techniques such as data flow diagrams. These diagrams can be used to graphically depict the output and input of the businesses processes. From these diagrams a data dictionary will be developed. During this phase structured decisions are analyzed. After analysis is complete the analyst prepares a systems proposal of the current system. If management chooses a recommendation in the system proposal then the analyst will proceed in that direction for the development of the system.

The next phase of the SDLC is the design of the recommended system. In this phase the analyst makes sure that data going into the information system is correct. Good form and screen design also play a role in this phase as they are important for effective input. This is all a

part of the logical design of the information system. Further, the HCI will be designed in this phase. Files and databases will also be designed at this point. Output design for screen or printed will be designed in this phase. Controls and backup procedures for data protection will be included in this phase as well as specifications for the programmers, which according to Kendall and Kendall include, "input and output layouts, file specifications, and processing details; it may also include decision trees or tables, data flow diagrams, a system flowchart, and the names and functions of any prewritten code routines" (2008, pp. 13).

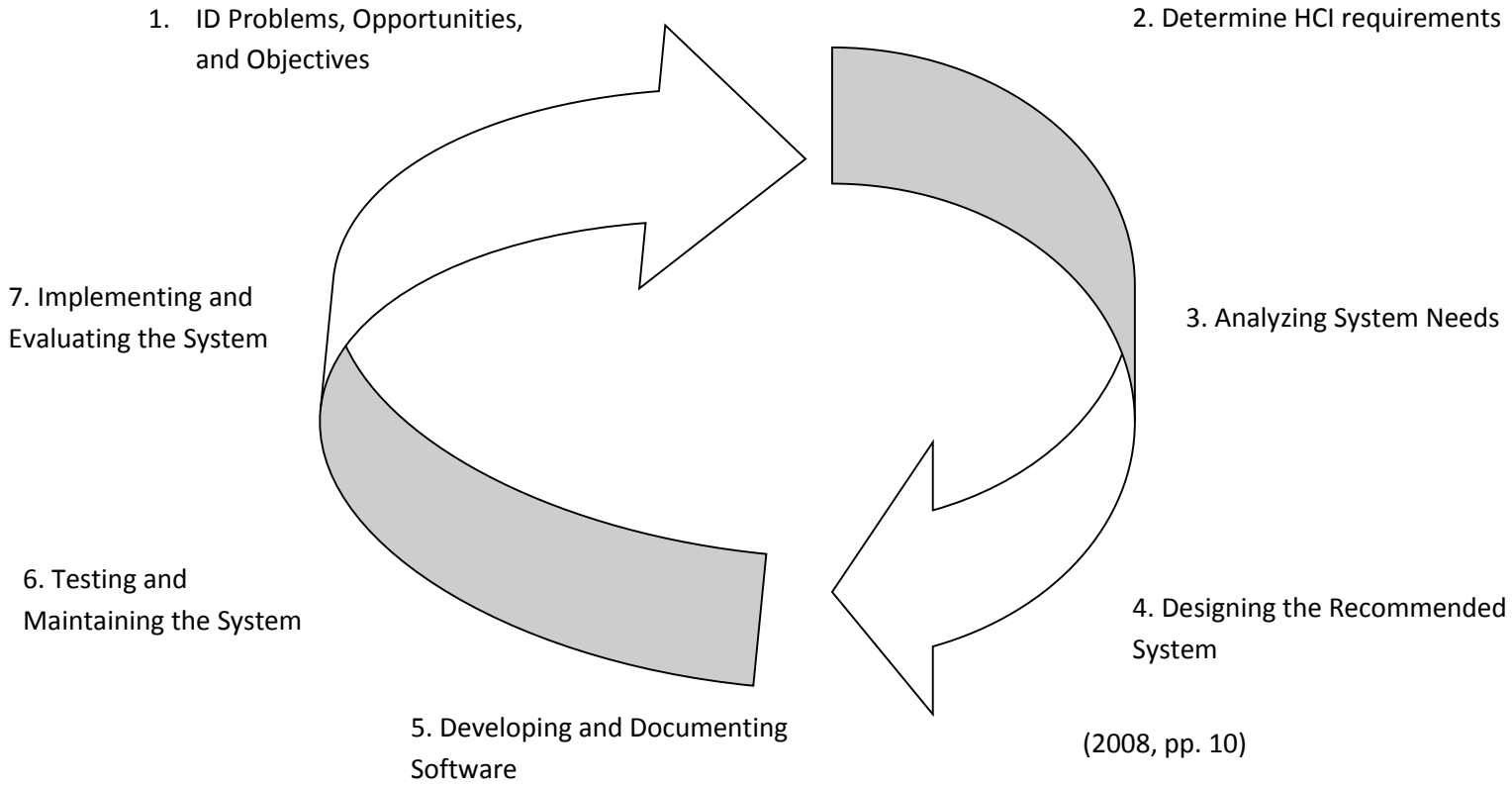
The next phase is developing and documenting the software. In this phase original software is developed with the help of the analyst. Kendall and Kendall detail that documentation for software, procedure manuals, online help, FAQs, and readme files are all developed with the help of users by the analyst (2008, pp. 14). During this phase programmers design, code, and debug the software. This can include another team of programmers who are walked through the code to ensure quality.

The next phase is testing and maintaining the system. Programmers alone or teams of analysts and programmers run tests with sample data to find problems then after this is done live data will be used. Maintenance of the system begins here and is continued throughout the lifecycle. Orderly procedures that the analyst uses throughout the lifecycle can help minimize maintenance in this phase (Kendall, Kendall, 2008, pp. 14). Also, some maintenance can be performed by the vendor trough updates via an internet connection.

The next phase is implementation and evaluation of the system. The analyst helps train users and some is done by the vendors. However, training is mainly the responsibility of the

analyst. Now is the time for the analyst to plan for a smooth conversion to the new system. According to Kendall and Kendall this includes, "converting files from old formats to new ones, or building a database, installing equipment, and bringing the new system into production" (2008 pp. 14). Evaluation takes place throughout the entire process of the SDLC and occurs at this point as a final note about the entire implementation process. A key point in the final evaluation process is to ensure that the intended users are using the system. Finally, Kendall and Kendall tell us that the system analyst can return to any point in the cycle as a result of a discovery of an error (2008 pp. 14).

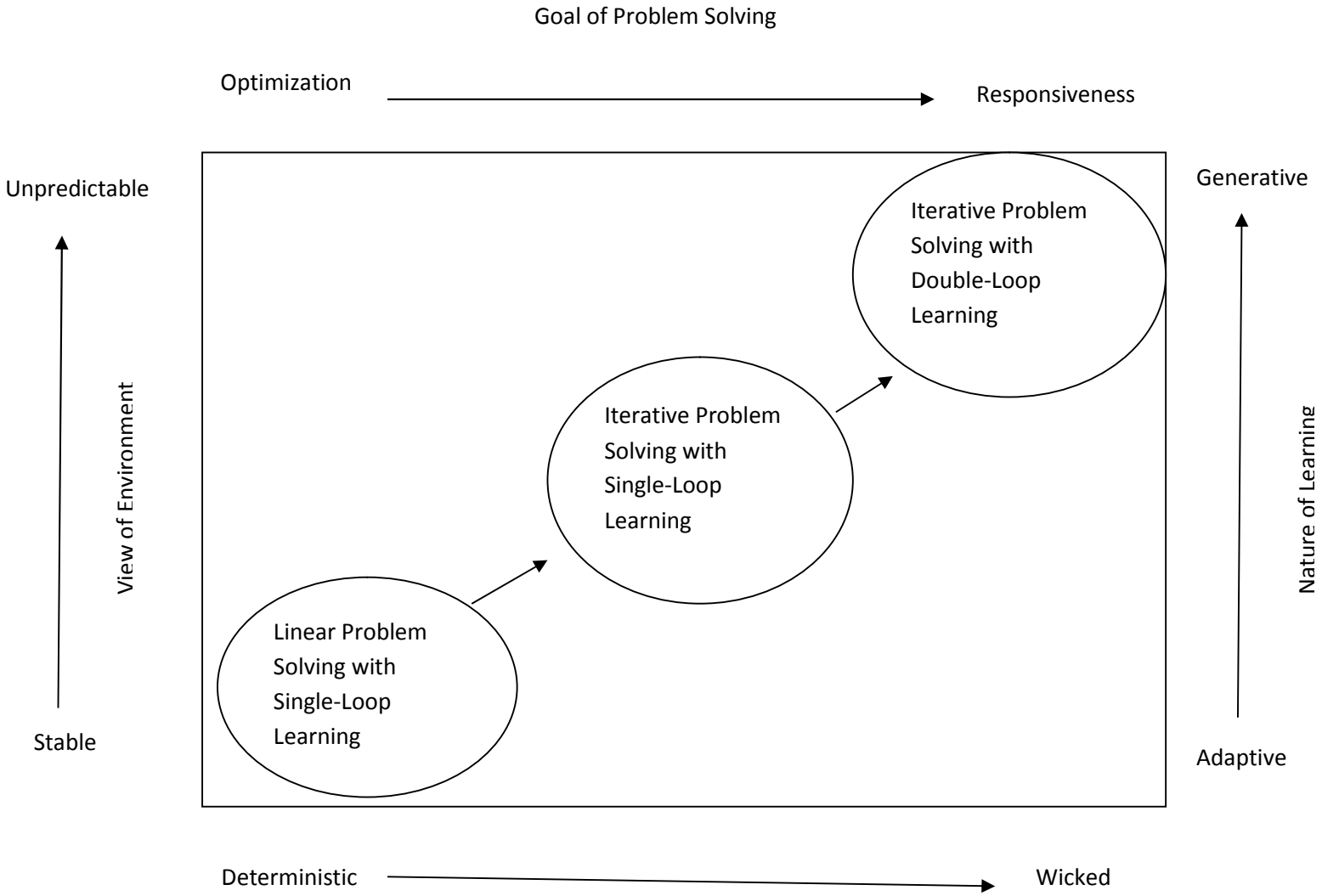
The following is a diagram of the SDLC according to Kendall and Kendall.



(2008, pp. 10)

This diagram shows the cyclical nature of the SLDLC. As you will see with other models they are not cyclical phases that integrate with each other but have their own unique characteristics.

A new methodology that is becoming increasingly successful and reportedly effective is Agile modeling. In Agile modeling project management is very important. Kendall and Kendall state that defining a system plan quickly, releasing software and developing it in a quick manner, and revising the software continuously to add new features are all parts of agile project management (2008, pp. 185). Programmers, analysts, and designers that work independently with agile methods can get great results. But the agile methodology brings a new strategy to the table, paired programming, which has produced outstanding results. The Agile methodology consists of Agile Core Practices, Agile Activities, Agile Values, and Agile Resources. Agile methodology is an evolution from previous methods such as the SDLC. Nerur and Balijepally give of this depiction of the evolution to the Agile methodology on the next page.



(2007, pp. 81)

There are four values of Agile Modeling. Communication, Simplicity, Feedback, and Courage are the values. Excellent communication in this value is the key. Since various types of

problems can arise during projects such as miscommunications (people may not understand others specialized industry jargon), communication barriers between people, (barriers such as programmers not wanting to leave their comfort zone of working with a computer all of the time and being a silent party during teamwork as a result of their displeasure.), and a culture that does not condone the spreading of new ideas and is stuck in one gear all can disrupt communication. Therefore this value focuses on communication to make sure that it is working in an excellent manner to induce the highest productivity. Paired programming is a part of this value as it facilitates communication between programmers on projects to boost efficiency. The next value is simplicity.

Simplicity has a certain jist to it. The jist is to not be overwhelmed by the entire project once it is put in front of you. To follow this value the analyst will begin with the simplest thing that can be done. This way, the simplest thing that we do today can be changed a little bit tomorrow (Kendall, Kendall, 2008, pp.186). This keeps project goals a reality and priority and facilitates efficiency in the development process. The next value of the agile methodology is feedback.

In the context of the agile mode feedback has to do with time. Feedback that is timely is the key and produces efficiencies in the development of the project. Whatever the time span of the feedback, if it is in a timely manner, that is the value that the analyst should strive for (Kendall, Kendall, 2008, pp. 187). A good example of this is one presented by Kendall and Kendall in the text. A programmer hands another programmer a program that has just been tested and the code is not feasible for the project hours before the project is due. Due to the

timeliness of this feedback the code can be corrected and the project can remain on track (2008, pp. 187). Feedback is also important in its normal context. Feedback can come from customers to update the analyst on any changes that need to be made. This type of feedback is very useful in keeping the project on track and keeping the customer involved in the decision making process. The next value in the agile methodology is courage.

Courage in agile programming lets the programmer focus on the task at hand instead of a deadline or any other distraction that may cause them to rush to get the programming done without it being correct. This courage enables focus and increases efficiency. Kendall and Kendall tell us that responding to feedback is courageous. It is risky but rewarding because it provides a hotbed for new ideas constantly being shared. This includes trust of teammates to get the job done (2008, pp. 187).

The agile principles consist of providing rapid feedback, adopting simplicity, changing incrementally, embracing change, and encouraging quality work. Providing rapid feedback helps the team stay on track and accomplish the tasks that are necessary for the project to be successful. Adopting simplicity is something that all programmers utilizing the agile methodology should adopt in order for the process of development to be successful. Everybody gets anxious to tackle big projects and get started right away and just dive into everything. This is a temptation that the agile programmer must resist. The agile method calls for starting with something simple not big and exciting. The reward is less stress, which helps the programmer focus on the goals that the customer has put in front of them. Since focusing on the customer is an important part of agile programming this works out nicely in the

methodology that is meant to respond with agility to market demands. Changing incrementally is an important part of the agile methodology. Kendall and Kendall tell us that the agility comes from incremental changes that are made in the smallest possible way that makes a difference. This means that incremental changes will impact code, the team, and the business requirements. They will all change incrementally beyond the product release date (2008, pp. 188). Embracing change is the fourth basic principle. Since teams need to respond with agility it is important that they are open to new ideas that will have a positive impact on their development or they may be ideas that the customer just has and wants integrated into the program. Whatever the case may be, to be agile is to be open minded, which relates to one of the agile values of feedback. Accepting feedback is being open minded. These two can be very powerful when combined. New ideas are generated and innovation takes place. The last principle is encouraging quality work. Doing quality work is going to help in the itineration process of the agile methodology. If the work is innovative and functional the first time then no bugs will have to be fixed in the next itineration and the program will continue to advance and be more competitive in the face of highly competitive market demands. Those are the basic principles of the Agile methodology. Kendall and Kendall identify some other that can help in certain situations. They include the mandate to teach learning; encouragement to make a small initial investment so that the good, not extravagant, work is done; play to win, don't play to avoid losing; and use concrete experiments to test the work that is being done (2008, pp. 189). Kendall and Kendall also show some other concepts that support the agile approach. They include using open and honest communication without fear (which supports the value of feedback); working with people's natural tendencies; claiming responsibility for a task rather

than ordering others to do something; locally adapting the approach you are learning for agile development and seeking to use honest measurement that doesn't pretend a preciseness that doesn't exist. Other principals include "model with a purpose", "software is your primary goal" and "travel light" (a little documentation is good enough) (2008, pp. 189).

There are also activities, resources, and practices in agile modeling. The four basic activities that are used in agile modeling are coding, testing, listening, and designing. Each one of these requires the analyst come to a balance with the demands of the project. Coding is considered as an important activity in the agile model because it helps develop new ideas. It is a language of its own and its abstraction can generate new ideas when viewed by an individual other than the original coder. This stimulates development; hence its importance as an activity in Agile methodology. Testing is the next basic activity in the Agile methodology. With coding comes the need for testing. With extreme programming comes the reliance on testing in order to deliver quality work. Since Agile modeling relies on automated tests, updates of the tests will be occurring during the project as well as the use of large libraries of tests. In Agile modeling things are done incrementally, so testing each increment is important for the success of the project in the long run and the delivery of quality work in each increment. The next activity is listening. With paired programming listening becomes very important for the pair to deliver the excellent work that paired programming is known to produce. Agile modeling stresses less documentation, so this makes listening very important in order to be able to meet project goals and not get bogged down in confusion as a result of not knowing what is to be done. Another important aspect of listening is its importance to the developer's relationship with the customer. The customer's participation in the development process is a key and it is

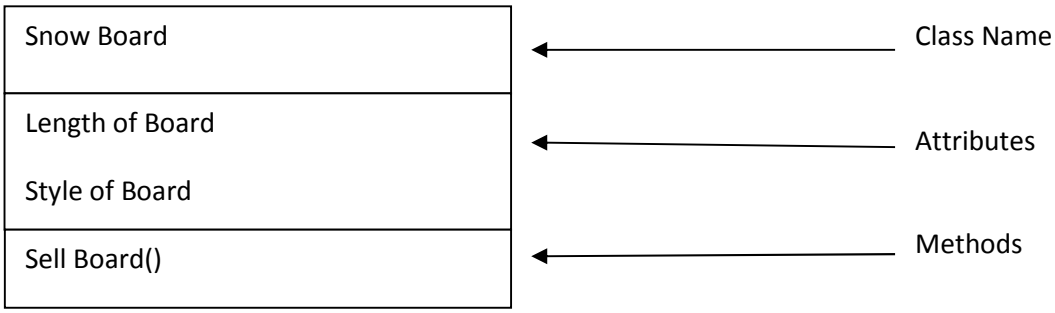
important for the developer to listen to the customer's needs, expected results, and goals that have been devised by the customer. Designing is the fourth basic activity in the Agile model. Design should be simple and flexible. Simplicity is an Agile value and principal and also works very nicely with the Agile principle of changing incrementally. Being flexible is part of the Agile principle of embracing change. Since systems are always evolving, this mindset helps keep complex systems development projects easy, which makes things more efficient and goals easier to focus on. Resources of Agile modeling include time, cost quality, and scope. They are viewed as being variable and not static. The Agile model deadlines are not meant to be extended and increments of the project are meant to be finished on time. This means that the most important parts of a project will be delivered to the customer first and the rest of the project will be added incrementally following the initial release. Cost is the next variable. Cost can be increased or decreased as needed in a project to fit work with the rest of the variables to deliver the product that is needed. Quality is another variable. Kendall and Kendall show us that quality is internal and external. Internal quality can involve testing software for functionality such as does the program do what it is supposed to do? External quality revolves around the customer perspective. Does the program act bug free (2008, pp. 88)? Scope revolves around the stories obtained by the customer. Once a story is obtained the analyst is able to adjust the project's scope according to the customer's needs. Adjusting scope can allow for the most important aspects of a project to be completed first in order to meet time deadlines. The Agile practices in modeling consist of four core areas. Short release time means that the most important part of the project will be released first followed by the rest incrementally. A forty-hour work week means that developers will be well rested. Because of

this the project quality will not suffer. An onsite customer provides expertise of the business, its needs, provides user stories, and defines goals. Paired programming is the last core practice. According to Kendall and Kendall, "Pair programming saves time, cuts down on sloppy thinking, sparks creativity, and is a fun way to program (2008, pp. 190)".

Systems can also be designed using Object-oriented approach using the unified modeling language (UML). According to Kendall and Kendall Object Oriented analysis and design facilitates logical, rapid, and thorough methods when creating new systems. The technique works well with systems that are undergoing continuous maintenance, adaptation, and redesign (2008, pp. 709). Object-oriented systems consist of objects, which are a part of classes. The reason behind this is to simplify the process of designing the information system. In Object-oriented design the object can be reused once it is a part of a class and the class defines the objects characteristics. This allows the reuse of coding, therefore saving time and speeding up the project. The concepts of Object-oriented design are objects, classes, and inheritance.

According to Kendall and Kendall objects are," Persons, places, or things that are relevant to the system we are analyzing. Objects may be customers, items, orders, and so on. Objects may also be GUI displays or text areas on the display (2008, pp. 710). Classes are meant to facilitate reuse of data. When an object is a part of a class it holds the characteristics of the class. For example, snowboards all may have the same type of design, like those used for graceful carving of the mountain and length of board, but the value for each snowboard may be

different. We can see how a class looks by using a simple example of a snowboard shop selling boards.



From this example we can also see that Object-oriented design consists of a structure (class) that contains the class name, attributes and methods. This is the basic unit of analysis in object oriented analysis and design. To further explain, according to Kendall and Kendall, "An attribute describes some property that is possessed by all objects of the class"(2008, pp. 711) and a method "Is an action that can be requested from an object of the class" (2008, pp. 711). The snowboards can come in all different lengths to make them suitable for their rider. All snowboards have this attribute. Since the store is in the business of selling snow boards, a method that can be called upon for the snowboard is for it to be sold. Inheritance is a concept of object-oriented systems. Inheritance is when a class has a child and it becomes the parent. The child (derived class) contains all of the attributes and behaviors of the parent (base class) and some new attributes and behaviors. This allows for less programming as the child can be defined with an already existing class. In terms of use with a database, the parent table is already in third normal form, which saves much time normalizing data.

Object-oriented think uses some unique techniques to spur the development process of the information system. CRC cards and what is referred to object think are utilized. These methods are used to create classes and objects. CRC stands for class, responsibilities, and collaborators. According to Kendall and Kendall, "CRC cards are used to represent the responsibilities of classes and the interaction between the classes" (2008, pp. 712). Here is how CRC cards work. The cards are created based on scenarios derived for the information system requirements. Sometimes analysts use them in a group, so small note cards will be feasible, otherwise they are created on a computer. The cards are used in a CRC session. This is a session where all of the analysts brainstorm to find all of the classes that they can. Once the classes are identified they are weeded and placed on a card and given to each person in the group. Next, scenarios are created and from the system requirements used earlier. The group then decides which classes work with what system functions. Analysts hold their cards in the air when it works with a particular function. Once in the air the card is considered an object.

The Unified Modeling Language (UML) provides a set of tools utilized in the documentation of the object-oriented methodology. Things, relationships, and diagrams are what UML is composed of. Things are classes, use cases, etc. that are used to create models. This helps describe relationships. There are behavioral things, group things, and annotational things. Behavioral things describe how things function. Group things create boundaries. Annotational things are used to make notes. Relationships are used to tie things together in the diagrams. There are structural relationships and behavioral relationships. Diagrams that are used in UML are either structural or behavioral. The structure diagrams are used with classes and how they relate to one another. Behavioral diagrams are used to describe how the

system is used. According to Kendall and Kendall UML tools are used to break down the system requirements into use case models and object models (2008, pp.745). Kendall and Kendall also outline these six steps in system development.

1. Define the use case model
2. Continue UML diagramming to model the system during the systems analysis phase.
3. Develop the class diagrams.
4. Draw statechart diagrams
5. Begin systems design by refining UML diagrams, and using them to derive classes and their attributes and methods.
6. Document your system design in detail.

(2008, pp. 745-746)

The object oriented design methodology is iterative, but it's characteristics make it unique for developing certain systems that have unique problems.

The differences between the three models are evident. Spangler, Hartzel, Gal-Or, and May discuss the Object-oriented programming in comparison to other models. Object oriented programming is incremental as is the Agile method. However, the impact on architecture when a change is made can be hardly even noticeable with the object oriented method (2009, pp.76). This makes the difference between the object-oriented method and the SDLC even more black and white. In the SDLC if you may have to scrap large portions of a project if they are not the correct solution to a problem. With object oriented it can be as simple as using new objects with preexisting classes or creating a new class. This means much less work. This is also

different from the Agile method, which stresses courage to not be afraid to scrap large amounts of code and start over again. Once again the object-oriented method architecture seems to be advantageous when working with large complex architectures that are constantly in a state of change as large amounts of the project will not have to be scrapped as a result of incremental and reusable design.

The Agile method can be compared to structured methods such as the SDLC by comparing seven strategies that can improve knowledge work discussed by Kendall and Kendall. These strategies are reducing interface time and errors; reducing time and effort to structure tasks and format outputs; reducing nonproductive expansion of work; reducing data and knowledge search and storage time and costs; reducing communication and coordination time and costs; and reducing losses from human information overload (2008, pp. 197).

Reducing interface time and errors occurs during system analysis, design, and development. The structured approach used many procedures and forms to function. Procedures such as classification of what type of software everyone must use and forms are used in documentation to allow other programmers to continue another programmer's work. In the Agile method, paired programming comes into play. Paired programming reduces time and errors and boasts an improved interface since the two programmers are both committed to creating a quality product and the programmers are working on the same code. Reducing the process learning time and dual processing losses focuses on all of the things that become cumbersome to the project. Kendall and Kendall state that these include programmers that do not have any experience being out into a project where they have to be taught and learn as the

project progresses. This slows progress down. The analyst may also have to learn how to use CASE software or a computer language. Documentation is also a considerable (2008, pp. 199). The Agile method boasts the ability to develop a system without CASE and significant documentation making it faster and more adaptive than a structured method. To reduce the time and effort to structure tasks and format outputs the Agile method uses short releases. The most significant part of the software is released first followed by more releases that add additional functionality. Kendall and Kendall state that a structured approach calls for the use of CASE tools; diagrams, project management software, detailed job descriptions; using and reusing forms and templates; and reusing code written by other programmers (2008, pp. 199). Since the Agile method is using less to accomplish its objectives it is an attractive alternative to a structured approach such as the SDLC. The Agile method reduces the nonproductive expansion of work. Since it uses short timelines to deliver results there is no problem with non-productivity. Structured methods such as the SDLC approach timelines differently. Since the timeline in the SDLC is viewed as being far off there is going to be manipulation of deadlines to make them longer shorter in a nonproductive manner. To reduce the data and knowledge search and storage time and costs the Agile method stays close to the customer. The customer is always there to guide the analyst throughout the development process. According to Kendall and Kendall, structured methods such as the SDLC are using questionnaires, STROBE, and a sampling plan (2008, pp. 200). All of these techniques consume considerable time during the development process vs. having the customer on hand to provide immediate information. To reduce communication and coordination time and costs the Agile method is not focused on tasks as the structured method is but focused on time. Short releases are the mainstay of the

Agile method. According to Kendall and Kendall the tasks are divided into groups in the structured method allowing a decrease in time communicating. However, division of individuals into groups can introduce errors (2008, pp. 200). To reduce losses from human information overload the Agile method uses a 40-hour work week. Keeping programmers fresh means better work. Structured methods keep analysts in a vacuum from the customer in order for work to get done on time.

After review of these methodologies once can come to the conclusion in asking the question if one of the models stands out as truly the best or does each model have its own advantages for certain situations. The conclusion is that each model has its own advantage for the situation at hand. For example, the object-oriented approach is best for systems going through constant change because of its ability to cut down on coding and development. The Agile method is great when a customer will be satisfied with a product that is not fully functional but the main characteristics of the program are satisfactory and the rest can be rolled out later. The SDLC is best when there is time for a thorough systems development process or when the organization is not open to change such as one that the Agile method brings with it.

References

Kendal K., Kendal J. (2008). *Systems Analysis and Design.* Upper Saddle River, NJ: Pearson Education.

Nerur S., Balijepally V. (2007). Theoretical Reflections on Agile Development Methodologies. *Communications of the AC, 50*(3), 79-83. Retrieved from Business Source Complete.

Spangler W., Hartzel K., Gai-or M., May J. (2009). Modeling Complexity in Physically Distributed Object-Oriented Systems. *The Journal of Computer Information Systems, 50*(1), 74-84. Retrieved from Proquest