The Office of the National Coordinator for Health Information Technology

Audio Transcript

Installation and Maintenance of Health IT Systems Software Development Life Cycle (SDLC) Model

Health IT Workforce Curriculum Version 4.0/Spring 2016

This material (Comp 8 Unit 5) was developed by Duke University, funded by the Department of Health and Human Services, Office of the National Coordinator for Health Information Technology under Award Number 1U24OC000024. This material was updated in 2016 by The University of Texas Health Science Center at Houston under Award Number 90WT00006.

Slide 1

Welcome to Installation and Maintenance of Health IT Systems, Software Development Life Cycle. This component Installation and Maintenance of Health IT Systems covers fundamentals of selection, installation, and maintenance of typical Electronic Health Records (EHR) systems.

This unit, Software Development Lifecycle, will discuss methods for planning for the creation, development, implementation, and eventual phase-out of software packages using various Software Development Life Cycle Models.

Slide 2

The Objectives for this unit are to:

1. Define the steps of the Software Development Life Cycle, or SDLC, and the purpose and importance of each.
2. Describe different models of the SDLC and their key differences.
3. Describe how and why an HIT software application would go through the SDLC.

The SDLC is a well-developed concept from the IT world that promotes an organized, long-term view of the software you're working with, from its "birth" to its "death" (hence the term "lifecycle").

It's important for those who work in healthcare IT to understand this model and apply it when appropriate. This will be crucial if you work in an institution that chooses to build its own EHR components. But even if your institution lets a commercial vendor make all the changes to the software, it will be helpful to understand the conceptual phases they are using … particularly since your institution's success will be dependent on the outcome.

We'll start by describing the SDLC and its importance, then we'll discuss the conceptual phases of the lifecycle. Then we'll look at three different models – the waterfall, iterative, and spiral models – that illustrate different views of the relationships between the phases. Then we'll go through an example of the SDLC in practice. Finally, we'll close with more remarks about the role of the SDLC in EHR systems.

Slide 3

The SDLC is a term used for modeling a detailed plan for the creation, development, implementation and eventual phase-out of a software or software system package. It's a complete plan outlining how the software will be born, raised, and eventually retired from its function.

Many different SDLC models exist. Each of these models was designed to fit a specific business needs model, to accommodate available resources and skills, or to take advantage of a specific programming language or toolset that would be used.

Usually, these models can be divided into two categories - the waterfall model and the iterative model - each employing a different workflow philosophy.

Slide 4

So why is SDLC important, anyway? Well, as computers and software became integrated into the business environment and businesses became more dependent on computers not only to manage their business data, but also to assist or track every aspect of the workflow process, it became increasingly apparent that poor design, or failure, of software can be quite costly in terms of lost productivity. Additionally, poorly designed software can increase security risks and decrease data integrity. Replacement of outdated or inadequate software can cost many thousands of dollars.

Therefore SDLC was designed to control the development environment to help ensure that developers produce a high quality system that meets or exceeds their customer expectations, is completed within time and cost estimates, works effectively within the designed infrastructure, and is inexpensive to maintain and cost-effective.

Slide 5

Factors for developing a successful SDLC are not unlike those already discussed in previous units for developing your successful project plan or for selecting your EHR system. Again, these factors are not steps in your SDLC; rather, they are elements that will dictate whether the SDLC will be followed, which in turn assures the success of the program being developed.

1. Management support - Developers need to have the support of the management as much as possible, since management will dictate the business need, budgeting, and top-level buy-in for the product.
   Technical and business expertise – As in any field, there are experts (in this case, programmers) who just know what needs to be accomplished even when the objective is originally presented. These programmers know which SDLC model is most appropriate for the programming language or toolkits that need to be utilized to ensure the software project will be successful. Likewise, business experts are also critical in the software development cycle, since they understand the overall demand and needed functionality for a particular software. Additionally, business experts can help determine whether the software will eventually show any cost savings over other products or processes.
2. Determining the product focal points - Some parts of the program should be rated a higher priority than other parts. Choosing which elements are most important will allow developers to make decisions when issues arise which may compromise the software's overall functionality, ensuring that there will always be some strong selling points in the developed software compared to a product that provides only mediocre service.
3. Follow well-defined procedures - Developers should have a clear understanding of goals at each phase, along with the methods and accepted tolerances for evaluating each of the goals.

4. Develop proper documentation for maintenance – Developing good documentation will help with the implementation and continued success of the product throughout its entire life cycle.

Slide 6

Now let's take a brief look at how these phases can relate to each other, initially using the so-called "Waterfall" model.

The initial assessment of feasibility is followed by an analysis phase, which is followed by the design phase, which is followed by the implementation phase, then the testing phase, then the maintenance phase.

Slide 7

Contrast that with this "iterative" or "incremental" model, which starts with initial planning and research. Then begins a cycle -- consisting of planning, requirements, analysis and design, implementation, testing, and evaluation -- which repeats as needed until the decision is made to do deployment.

We'll discuss the models in more detail at the end of this Unit. Now let's discuss what each phase generally entails.

Slide 8

Initiation

In the software vendor world, where profits are realized by fulfilling consumer market needs with new software products, initiation is where a need is identified for a new software system. Software development companies use this stage to determine the needs of the present market. The software vendor's management is often involved in this stage as they want to determine what the developers have to do and how it will impact the market.

In a clinic or other healthcare institution, this need is usually identified by clinicians or staff such as a flawed workflow process or other issue.
For instance, a healthcare clinic currently uses three different programs to record patient data, dispense online prescriptions, and run the business office, requiring a lot of work overlap and generation of paper documentation between systems. Both the physicians and the billing department are looking for a more efficient way to communicate and improve efficiency. They would like a system that can communicate seamlessly between the various business components and streamline operations.

A Project Manager typically would be assigned and would eventually generate a Concept Proposal – a document which identifies the problem and why the new system needs to be pursued. Upper management would then review the proposal and approve it, and the project moves on to the next phase.

Slide 9

The Concept Development phase begins when the Concept Proposal has been formally approved but when additional study and analysis are required prior to system development activities.

We begin to analyze what will be necessary to complete the project. Here all relevant factors are analyzed to develop the project scope.

Several reports can be created here:

- Feasibility Study; in other words, will it work?
- Cost/Benefit Analysis; in other words, is the cost really worth it?
- System Boundary; in other words, how far should the project go?
- Risk Management; in other words, what will happen if we don't do it?

These reports are then presented to the powers that be, and a decision is made whether or not to go ahead. They approve the funding. If they give their approval, it's on to the next phase.

Slide 10

During the Planning phase, we determine who is doing what, when, and how, along with the personnel and other resources that will be needed to complete the project. For instance, should we use existing personnel or hire consultants? During this phase we determine what developmental resources will be required to create the specific software.

Other questions are also contemplated during this phase:

- Should we develop in-house software, or buy it off the shelf?
- What are the "deliverables" – such as completed software programming and documentation, user manual, training, testing plans, etc.?

Finally, a planning document is submitted to management for approval.

Slide 11

The Requirements Analysis Phase focuses on what the system will do, in an effort that considers all stakeholders, including sponsors and potential users, as important sources of information.

During this phase you will determine what Operating System, or OS, and interfaces are required; e.g., will it run with Windows or LINUX? Here you will answer a number of other questions as well: What functionality will be required? Should it be run with the mouse, keyboard, or touchscreen input? How much training will be required of the user? Will a new room be needed for the hardware that runs the system?

There are a variety of techniques that can be used to gather the requirements, but some key points to remember are that the requirements must be systematic, verifiable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Once the requirements documentation is approved, the project moves on to the design phase.

Slide 12

The Design Phase takes those requirements and develops a detailed blueprint outlining the software specifications. Here, the program architecture, which defines the various software components, along with their interfaces and behaviors, is established.

The design phase is also where much of the program documentation begins to take shape, including the Maintenance Manual, Operations Manual, and Training Manual.

Flaws in the original planning, which require some adjustment, are often revealed during the design phase.

Slide 13

The system is built during the development phase. This includes source-code files, header files, make files, and binaries.

This is where everything is coded and assembled, and the actual design of the system is realized as “living, breathing” software. This process is usually a team effort with its own set of sub-goals and milestones, involving many software developers who coordinate their efforts to realize a final product.

Slide 14

Integration and testing of the entire system is a formal, documented testing procedure, which ensures that the system performs as designed.

Testing the roll-out of the system also begins, including migration of existing data into the new system, as well as usability.

Usually, the system is rolled-out over a weekend so that if anything goes wrong, the old system is still active and available.

Integration and testing is a critical step. If the software fails testing, it cannot be trusted to work.

Approval of testing and test results is necessary before the project moves into the next phase: implementation.

Slide 15

Now that the software has been formally tested and passed, it is time for distribution to the production environment. In this implementation phase, user training takes place, previous data is migrated, and the system is brought online.

After the system “goes live”, it is once again tested and data is collected to determine if it is working to specifications. This process is often referred to as a “debriefing”. It is also where any problems that were not crucial to the implementation can be addressed and any necessary changes to the system documented for future versions.

Slide 16

The Operations & Maintenance Phase is the day-to-day operation of the software after deployment. The software must be maintained, patched, and regularly updated to ensure long, reliable life. Upgrades can be tested and deployed to improve functionality.

Slide 17

Every software application eventually becomes obsolete. Perhaps a new system is being developed, or this system cannot keep up.

Whatever the reason, disposition involves more than just shutting off the server. Often, the system may be kept going due to regulatory requirements or because there are still projects using it. Sometimes there is data on board that must be kept safe and secure even after its useful life is over.

Software disposition needs to be planned to ensure that you meet all regulatory requirements and ensure a smooth transition to whatever the replacement product will be. A disposition plan must be developed which may outline everything from the dismantling of the software itself, to the safe and secure disposal of old hardware and data, to archiving of documentation.

Slide 18

As briefly shown and discussed earlier in this presentation, many different SDLC models exist. Each of these models was designed to fit specific business needs, to accommodate available resources and skills, or to work with a specific programming language or toolkit.

Usually, these models are divided into two categories, the Waterfall Model and the Iterative model, each of which employs a different workflow philosophy.

Slide 19

The waterfall model uses more traditional planning, testing and implementation techniques to design and implement new software products. This model promotes strong documentation of each step of the development process.

The waterfall SDLC model represents a sequential development process, where progress is seen as flowing steadily downwards through each of the phases of development.

Winston W. Royce has been given credit for formalizing the waterfall model in an article around 1970, where he presented this modeling concept as flawed from a software development standpoint.

The waterfall development model is actually a product of the manufacturing and development industries where making after-the-fact changes is often prohibitively costly. Therefore progression to the next phase of development does not typically commence until the previous phase has been perfected and ”locked down.”

Many have criticized the waterfall model, citing that it is difficult, if not impossible, to finish a phase of a software product’s lifecycle perfectly before progressing forward to the next stage. To that end, several variations of the waterfall model have been created to help address some of these issues.

Slide 20

This diagram, which I described on an earlier slide, depicts just one of the many variations of the waterfall model employed by developers. As this variation of the waterfall model suggests, the waterfall model can best be described as a sequential software development process whose workflow progresses in a linear, downward fashion, just like a waterfall flows.

Slide 21

Using a waterfall methodology is most likely to be successful when the complexity of the system is low and requirements are static, but there is little room for mistakes and no process for correcting errors after the final requirements are released.

Feedback can be quite limited when using this approach. And, as I mentioned previously, most believe it is nearly impossible to finish a phase of a software product’s lifecycle perfectly before progressing forward to the next stage.

Slide 22

Iterative and Incremental model:

These models were developed in response to identified weaknesses of the waterfall model and often considered cyclical in nature. One variation of the iterative model is the Spiral model, which we will look at after we see a more basic Iterative model.

This approach works well in environments where perceived requirements are subject to change as the project progresses or where more feedback is warranted.

Let’s take another look at the illustration.

Slide 23

As you can see from this diagram, which I described earlier in the presentation, in iterative models, the phases of software development take on a more cyclical nature.

It starts with an initial planning phase and ends with deployment, with the cyclic interactions in between, or even after the initial deployment occurs. With this model, several deployment cycles of the product are possible as the software becomes further refined, analyzed, and tested, and as new enhancements are added.

Slide 24

Another variation of the cyclical model is the spiral lifecycle (or spiral development) model used in IT. This model combines the features of another model, called the prototyping model, with those of the waterfall model. The spiral model is intended for large, expensive, and complicated projects where client responsiveness is a significant issue.

In this model, one or several prototypes may be created throughout the lifecycle. At each iteration, the prototype is tested and further input is gathered until all concerns have been adequately addressed.

Let's walk through the diagram. The action begins in the center of the spiral, with initiation, and then cycles repeatedly through four conceptual quadrants: Identify, Design, Construct, and Evaluate. The first cycle contains the phases Concept and Risk Analysis. Later cycles contain the phases Implementation, Testing, and Delivery.

Slide 25

As you can see, the SDLC is very similar to a project plan, but it integrates several software-specific aspects into the planning infrastructure to specifically address the concerns associated with developing and deploying a new software system. However, you should consider an SDLC as augmenting your overall EHR project plan, not replacing it.

Employing SDLC in your environment, when developing a new EHR system or designing changes to an existing system, is crucial to ensuring that your new software, whether developed in-house or purchased off the shelf, adequately meets expectations while mitigating overall risk to the organization.

Slide 26

Now let's discuss a scenario in which EHR installation utilizes SDLC principles.

Sunny Happy Care (SHC) Clinic, a small primary care practice, wants to upgrade their paper records to an EHR system. Before purchasing and deploying, they do extensive planning, including evaluation of their requirements and analysis of market options. They ultimately select and implement a commercial EHR.

Slide 27

You'll have recognized those actions by the clinic staff as several of the general steps in the SDLC. The fact that they are specifically following an iterative SDLC model in their deployment soon becomes clear. According to the project plan, the business manager has been assigned to test and evaluate the EHR after go-live. In her analysis, she determines that the SHC clinic staff is spending excessive time manually entering laboratory data, since a laboratory integration module was not part of the initial purchase. Thus the staff enters a new cycle of planning, requirement-gathering, & analysis of vendor options, leading to purchase & deployment of a laboratory module. Subsequent re-testing & re-evaluation of the EHR now shows satisfactory improvement in staff effort & availability of lab data.

Slide 28

This concludes our unit on the Software Development Life Cycle.

In summary, the SDLC is a set of steps that were codified by the software development industry but are useful in executing many types of projects. Common steps include concept development, planning, requirements analysis, design, development, integration & testing, implementation, and operations & maintenance.

The relationships between these steps have been formulated into different models based on different needs. The waterfall is one such prominent model, and it emphasizes steady progress through the steps and careful completion of each before moving on. The iterative, or incremental, is another prominent model, and it emphasizes iterations, in which the steps are repeated in a cycle of improvement. It has variations such as the spiral.

The SDLC is useful for EHRs, and its steps should be carefully considered whether one is creating a new EHR or deploying a purchased EHR, in order to maximize the success of the project.

Slide 29 (Reference slide)

No audio.

Slide 30 (Reference slide)

No audio.

End.