**The Systems Development Life Cycle (SDLC) as a Standard:
Beyond the Documentation**

Dianne Louise Rhodes, Connect, International Inc., Rockville, MD

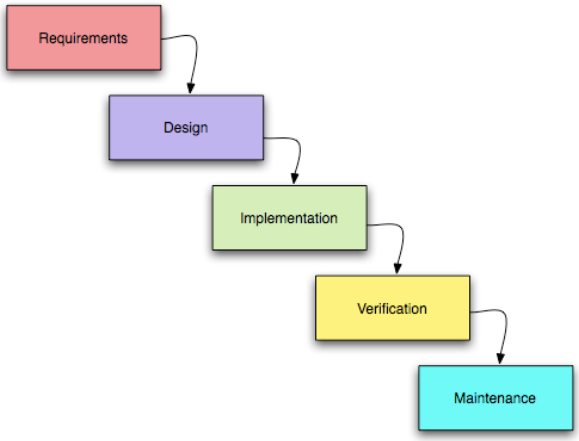
ABSTRACT

Has your company adopted the Systems Development Life Cycle (SDLC) as a standard for benchmarking progress on a project? Have they developed Word and other templates for documents created during SDLC? In three of my most recent positions, the stress was put on completing the documents according to schedule, rather than emphasizing the work. The work involved cataloguing requirements, analyzing them and developing a good design document, and thoroughly testing the resulting code. When I first started programming in SAS®, I was lucky to get any users requirements at all; it was always "I'll know it when I see it." But with the emphasis on the documentation, and not on the analytical work behind them, the project still falls behind schedule because of missed requirements. If the requirements are not thoroughly complete when coding begins, it is likely to fail in the testing phase, especially if the independent test team gets a better, more complete version of the requirements than the development team. We discuss the work that is involved in detail for producing sound requirements, design, and testing protocols. Consider the retirement of legacy software as part of the SDLC. Some useful templates (in Excel, perhaps) to help non-programmers specify the reports they want.

INTRODUCTION

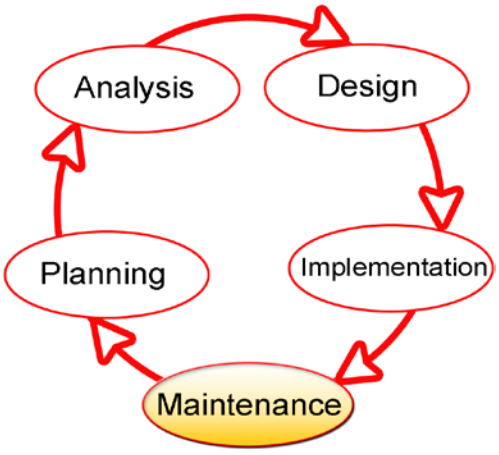
The exact naming and components of the phases can vary and are adapted to your company's specific needs and adjusted to the scope of the project (Fulton 2003).

PHASES



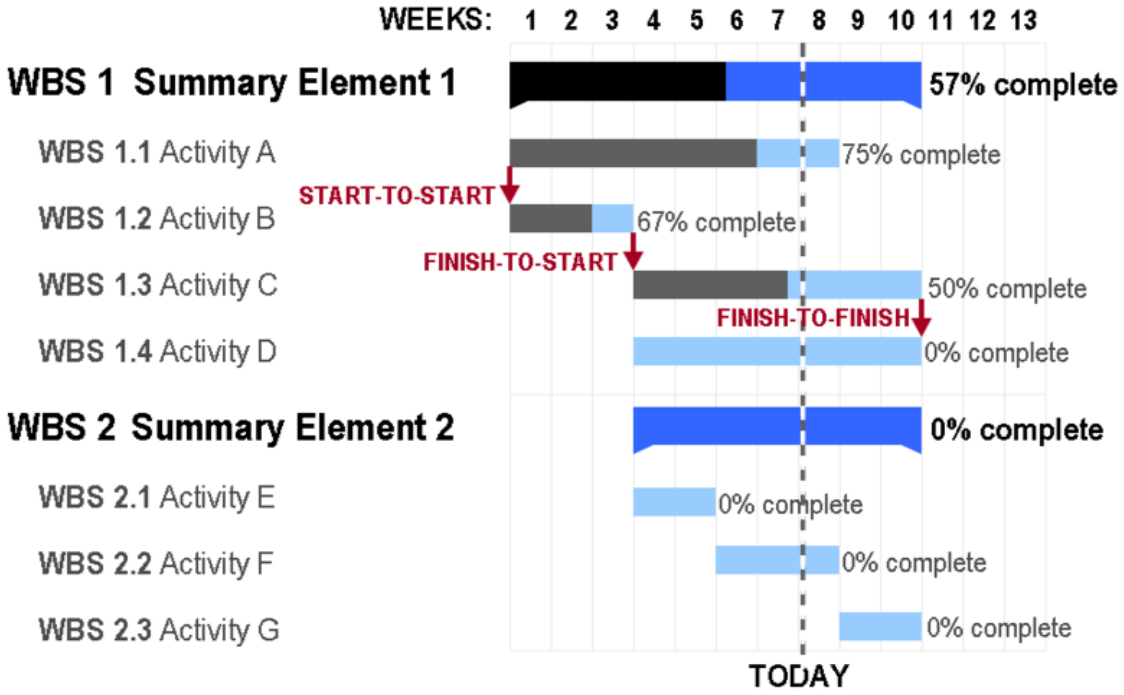
Traditional view of the SDLC is a waterfall approach. One phase keys off the end of the previous phase.

A more flexible and accurate description is of an interative process. No one ever gets it exactly right the first time and an iterative process depicts this.

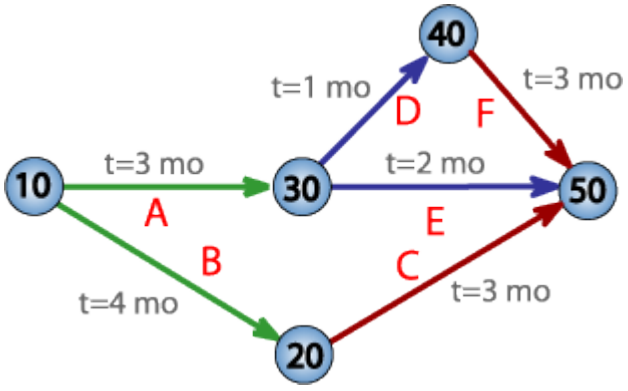


PLANNING

In this phase, the background and scope of the SAS and other programming to complete the project are estimated. This is done to determine the size and level of effort, staffing plans and budget. The project goals are set, and roles and responsibilities are identified. A project plan for performing the work and managing the project is developed. This should include tasks, schedules, and assignments. The project management will be implementing tracking and oversight mechanisms, configuration management activities, training, and a test plan / strategy. (Helton 2002) At this phase, documentation and validation that will be required are developed.



These efforts may be described as “proof of concept.” The job is to identify risks and risk assessment of assumptions and identify the critical path (CPA) of the work. The critical path is often identified using software such as Microsoft Project. Within a project the final project plan is often in the form of a Gantt Chart (using Microsoft Project or other software for projects of medium complexity or an excel spreadsheet for projects of low complexity).



The benefit of using CPA within the planning process is to help you develop and test your plan to ensure that it is robust. Critical Path Analysis identifies tasks which must be completed on time for the whole project to be completed on time. It also identifies which tasks can be delayed if resource needs to be reallocated to catch up on missed or overrunning tasks. The disadvantage of CPA, if you use it as the technique by which your project plans are communicated and managed against, is that the relation of tasks to time is not as immediately obvious as with Gantt Charts. This can make them more difficult to understand.

A further benefit of Critical Path Analysis is that it helps you to identify the minimum length of time needed to complete a project. Where you need to run an accelerated project, it helps you to identify which project steps you should accelerate to complete the project within the available time.

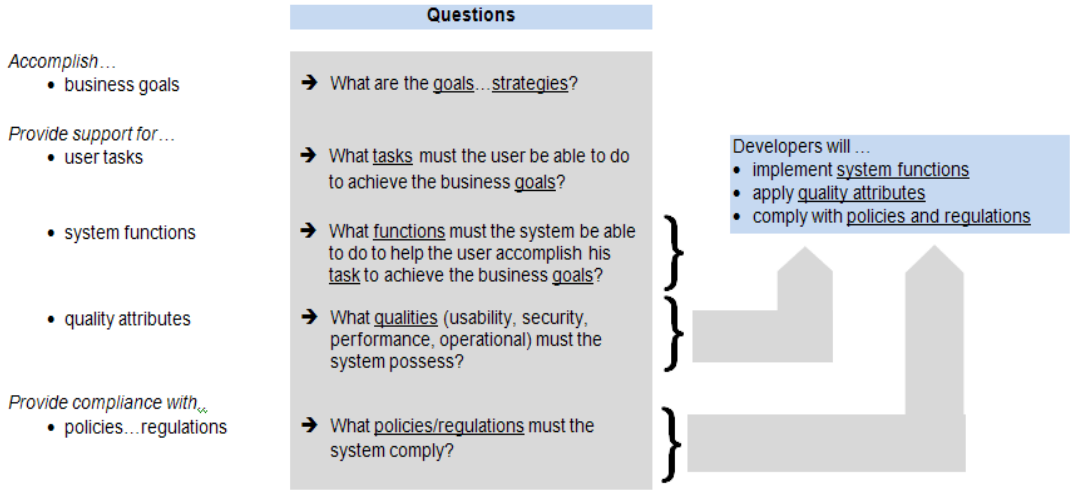
An example of failure to identify the critical path to success – Building the Denver, Colorado airport and not understanding that baggage handling was on the critical path. The airport's computerized baggage system, which was supposed to reduce flight delays, shorten waiting times at luggage carousels, and save airlines in labor costs, turned into an unmitigated failure. An airport opening originally scheduled for October 31, 1993, with a single system for all three concourses turned into a February 28, 1995, opening with separate systems for each concourse, with varying degrees of automation.

The system's $186 million original construction costs grew by $1 million per day during months of modifications and repairs. Incoming flights on the airport's B Concourse made very limited use of the system, and only United, DIA's dominant airline, used it for outgoing flights. The automated baggage system never worked as designed, and in August 2005, it became public knowledge that United would abandon the system, a decision that would save them $1 million per month in maintenance costs. (Wikipedia)

ANALYSIS – REQUIREMENTS. FOR CLINICAL TRIALS STATISTICAL ANALYSIS PLAN (SAP).

The requirements are developed by the end user of the software and not by the developer. There are a number of templates for this work. One is the Use Case developed by Ivar Jacobson. Developed in 1986, Ivar Jacobson, who went on to become an important contributor to both the Unified Modeling Language (UML) and the Rational Unified Process (RUP), first formulated the visual modeling technique for specifying use cases. Originally he used the terms usage scenarios and usage case, but found that neither of these terms sounded quite right in English, and eventually he settled on the term use case. Since Jacobson originated use case modeling many others have contributed to improving this technique, including Kurt Bittner, Ian Spence, Alistair Cockburn, Gunnar Overgaard, Karin Palmquist and Geri Schneider.

During the 1990's use cases became one of the most common practices for capturing functional requirements. This is especially the case within the object-oriented community where they originated, but their applicability is not restricted to object-oriented systems, because use cases are not object-oriented in nature.



These are functional requirements and sometimes are more of a wish list then requirements analysis. Rational Unified Process (RUP) is a package distributed by IBM. The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM. RUP is not a single concrete prescriptive process, but rather an adaptable process framework, which is tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. The product includes a hyperlinked knowledge base with sample artifacts and detailed descriptions for many different types of activities. RUP is included in the IBM Rational Method Composer (RMC) product which allows customization of the process. RUP is based on a set of building blocks, or content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved. The main building blocks, or content elements, are:

- Roles (who) – A Role defines a set of related skills, competencies, and responsibilities.
- Work Products (what) – A Work Product represents something resulting from a task, including all the documents and models produced while working through the process.
- Tasks (how) – A Task describes a unit of work assigned to a Role that provides a meaningful result.

Within each iteration, the tasks are categorized into nine disciplines: six "engineering disciplines" (Business Modeling, Requirements, Analysis and Design, Implementation, Test, Deployment) and three supporting disciplines (Configuration and Change Management, Project Management, Environment).

Unfortunately, the use of these tools is not standardized, and is subject to interpretation. Some shops may want to prepare report mock-ups. I have frequently seen these done in Excel. These could be part of the appendix to the requirements documents. The requirements should be a description in non-technical terms ("English") of the business rules being implemented. These are considered detailed functional requirements. There should also be a validation plan, which will help the testing team develop test cases and scenarios.

DESIGN

The Design document should reference what you are going to build to meet the requirements, and not how (Reap, 2005). This is described in broad terms: it can include pseudo code but shouldn't contain actual code functionality. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input. At this phase the test plans are developed. The level of review under the test plan depends on the level of risk to the project. The project gets system architect approval if it is going into a legacy system, to ensure that the changes are not going to "break" software already in place.

IMPLEMENTATION AND ACCEPTANCE

To launch the coding phase, develop a shell program that is then put under some form of version control, for example Source Control Management from SAS. This phase includes the set up of a development environment, and use of an enhanced editor for syntax checking. It is at this phase that development testing or unit testing occurs. Each developer insures that their code runs without warnings or errors and produces the expected results.

User Acceptance Testing (UAT) is a second part of the acceptance phase, which is ideally conducted by an independent test group. This includes the development of an Independent Test Plan put together by an Independent Test Team. Ideally these would be programmers, but often they are not. This phase verifies input/output and reviews the expected results. The Test Plan includes development of test data with test cases and scenarios which exercise all logical paths. The results are the validation of the code. This phase is also where regression testing and sign off occurs and the test team verifies that the development outputs still match the production outputs where expected.

PRODUCTION / MAINTENANCE

User's guides and training are developed to reflect any new functionality and changes which need to be identified to the production staff. Any changes needed to operations and/or maintenance need to be addressed. Every run in production needs to be verified. Any problems with production need to be addressed immediately. A Change Request system may be set up to allow for feedback for enhancements.

RETIREMENT

When the legacy system has been completely replaced, it is time to retire the system. Users are warned and explanations given of the new system. For example, we recently retired the use of FTP for Secured FTP. This required notification to the users, and tracking to find which users were still using the old system.

REFERENCES

Axelrod, Elizabeth (2009). "Boot Camp for Programmers: Stuff you need to know that's not in the manual." Proceedings of SAS Global Forum (SGF) 2009.

Brown, Rachel and Fulton, Jennifer (2008). "CSI: San Antonio – Common SAS Issues in Our Programs and Tips for Better Investigation of your SAS code." SGF 2008.

Fulton, Jennifer and Black, Stephen (2003). "Documentation and Accountablity: Why Your SAS Programming Projects Can Benefit from Implementing a System Development Life Cycle (SDLC)." SCSUG 2003 proceedings.

Helton, Edward, Halley, Patricia and Handelsman, David (2002). "SAS Solutions for Addressing 21 CFR Part 11 Compliance – the P21 Biomedical Knowledge Platform." 2002 PharmaSUG proceedings.

Howard, Neil (2003). "Beyond Debugging: Program Validation." SUGI 28 proceedings.

Newhouse, Russell (1997). "Validation and SAS Programming: Benefits of Using the System Life Cycle Method." SUGI 22 proceedings.

Reap, Chuck (2005). "A Regulatory Compliant Process for Developing SAS-Based Reports." SUGI 30 proceedings.

Wikipedia "Systems Development Life Cycle (SDLC) Life-Cycle Phases."

ACKNOWLEDGMENTS

I want to thank my colleagues at the Census Bureau for their support and timely feedback.

CONTACT INFORMATION

Comments, questions, and additions are welcomed.
Contact the author at:
Dianne Louise Rhodes
Connect International, Inc.
2275 Research Blvd., Suite 500
Rockville, MD 20850
Work Phone: (301) 763-2093
Email: diannerhodes@comcast.net

The tips database is an MS Access® database. If you would like a copy, contact me and I will provide it to you.

TRADEMARKS