Integrated Action Learning Project Documentation

Implementation of an Iterative and Incremental SDLC (Systems Development Life Cycle) Model Development Project for a Financial Services Organization

Jack Weintraub

TS4990 Integrated Action Learning Project

Instructor, Dr. Sharon Bender

March 3, 2006
# Table of Contents

1.0 Project Description ........................................................................................................ 3

2.0 Research .......................................................................................................................... 5
   2.1 Current Implementations of a waterfall SDLC ............................................................... 5
   2.2 Systems Project Utilizing an SDLC ............................................................................. 8
   2.3 Latest crop of iterative and incremental lifecycles .................................................... 9
      2.3.1 XP ......................................................................................................................... 9
      2.3.2 SCRUM ................................................................................................................. 9
      2.3.3 Crystal .................................................................................................................. 10
      2.3.4 RUP (Rational Unified Process) ........................................................................ 11

3.0 Analysis .......................................................................................................................... 12
   3.1 Gaps in waterfall SDLC ............................................................................................... 12
   3.2 Non-Waterfall Project Types ...................................................................................... 12
   3.3 Process Features Required for Identified Project Types ............................................ 12

4.0 Iterative and Incremental SDLC .................................................................................... 14
   4.1 Initiate Phase ............................................................................................................... 14
   4.2 Build Phase .................................................................................................................. 16
   4.3 Test Phase (System Test) ............................................................................................ 19
   4.4 Test Phase (User Acceptance Test) ............................................................................ 20
   4.5 Deploy Phase ............................................................................................................... 21

5.0 References ..................................................................................................................... 22
1.0 Project Description

The purpose of this IAL project is to plan, design, develop, and implement an Iterative and Incremental SDLC model for a Financial Services organization. Typically, the systems and maintenance work is developed using a standard waterfall development methodology. That SDLC (Systems Development Life Cycle) methodology advocates the serial execution of a series of project phases. During each phase one or more workproducts is created, each progressively defining and building the required project deliverables. This highly systematic approach provides a theoretically logical roadmap for technology personnel to follow in creating and maintaining systems. The same roadmap provides clarity for internal and external audit entities to verify that the technology personnel are faithfully following the standards. The overlying assumption is that system requirements are defined, analyzed, and frozen before design and subsequent development can begin. This assumption of frozen requirements and perfectly serialized execution is a reality for very few projects in the annual project portfolio of any Financial Services organization. Historically, there are two constants within the majority of systems projects in these types of institutions….the requirements (and scope) will change during the course of the project and the business wants the project delivered to production as soon as possible. These constants have a direct effect on the timeliness, cost, and quality of the system deliverables.

The goal of this project is to define and implement an SDLC model for a Financial Services organization that has the following attributes

- Embrace change throughout the life cycle
- Deliver value, often in small frequent increments
- Satisfy audit requirements for adequate procedures, evidence, and control
- Provide a delivery mechanism that is accessible by all technology personnel
- Provide a mechanism for facilitating overall governance against the SDLC model
2.0 Research

There were three major aspects to the research performed for this project.

- Study current implementations of a waterfall SDLC as it has been implemented at several financial services institutions.
- Identify and review different types of systems projects that would use an SDLC.
- Research the latest crop and most popular iterative and incremental lifecycle models used today.

2.1 Current Implementations of a waterfall SDLC

My research sample included four SDLC models from Citigroup Asset Management, JP Morgan Chase, The Bank of New York, and GE Financial Assurance. The Citigroup, JP Morgan Chase, and GE models were clearly straightforward, "one-size-fits-all" waterfall models. The Bank of New York model was the most flexible as it altered the SDLC and deliverables to project type, size, and budget. It wasn’t clear from the Bank of New York model what guidance was in place to determine what model would be used for a given project or what governance was in place to make sure the implementation was correct for that project. Graphical representations of the GE, JP Morgan Chase, and Bank of New York models are included below.
Citigroup Asset Management (derived from an SDLC Powerpoint Pitch)

Pre-Initiation (Tolgate)
- Business need is explored and documented in a business case.
- Project Manager wishes to explore implementing this improvement through technology is explored.
- High level solution is proposed and a high level project plan and cost estimates (Level 0) are developed.
- The pre-initiation tollgate must approve the feasibility analysis and agree to undertake definition work on the project.

Initiation (Tolgate)
- The project is set up in Citigroup.
- Project scope, communication, hardware/environment/architecture reviews are prepared.
- Project plan and cost estimates are updated (Level 1). STSat is required.
- The Architecture Forum tollgates the initiation review and approval.
- If the project is approved, a project team is assembled according to a Roles & Responsibilities matrix. The communication plan is established.

Definition
- The user defines business requirements.
- Good opportunity to introduce team (including business) to the structure and discipline of SDLC.
- Based on the business requirements document (BRD). Technology further defines the requirements and updates project plan and cost estimates (Level 2).
- The business and functional requirements are used as a basis for test/validation plans in future phases.
- Dependency is identified and documented.
- Bugs/Issues analysis is conducted.

Design (Tolgate)
- Based on the requirements, a detailed technical architecture and full design document are developed.
- The project plan is finalized, subject to later adjustments.
- The final (baseline) cost estimate (Level 3) is developed and approved.
- The COB plan is delineated
- In-depth design review is attended.
- The Information Security requirements are finalized in the design and STSat takes place.
- The Technical Design Review tollgates the design.
2.2 Systems Project Utilizing an SDLC

The following systems project types are candidates for and SDLC of some type.

- **Corporate Initiative** – These are projects with a firm set of requirements (for example, new desktop images, security patches, dbms upgrades, etc.)
- **Infrastructure** – These projects plan, test, and rollout infrastructure improvements to the computing, network, and telecommunications environment of the corporation. There are firm requirements for these projects. Often, infrastructure changes need to be configured differently across different regions (for national or global firms) and, thus, the implementation and testing activities may be iterative in nature until the proper configurations have been finalized.
- **Package Implementation (COTS)** – Even though these projects are based on an already developed commercial (or open-source) package there can be significant customization required. That customization is based on an initial set of requirements and iterative changes to those requirements as the project progresses.
- **New Development (home-grown app)** – These projects are initiated with an initial set of requirements and change is introduced in the form of new/evolving requirements
throughout the project. Changes may be caused by revised thinking about the associated business logic, competitive pressures, scheduling, or cost issues.

- **Maintenance** – These projects encompass small changes to functionality and/or bug fixes. They are relatively short projects in duration (one, possibly two man months). The scope of the maintenance release project can change due to many factors including new business requirements, project deadlines, etc.

- **Proof-of-Concept** – These types of projects are purely undertaken to test out the use of new technologies and configurations for future development/implementation purposes. The proof-of-concept projects have fairly firm requirements, may iterate as working prototypes are configured and re-configured, and have no technology deliverable other than a final report of the results.

- **Research** – Technology organizations conduct research projects into new technologies and platforms for future development and implementation. Unlike the proof-of-concept project, research projects do not necessarily touch the technology under research. These projects survey the “playing field” of technology, vendors, and competitive use of the area of interest.

### 2.3 Latest crop of iterative and incremental lifecycles

#### 2.3.1 XP

The XP team does not create formal requirements and design specifications. XP projects start with recording “stories” on index cards. The stories are placeholders for functionality, for instance a story could say, “Check for valid credit card number”. Developers on an XP team will derive further details on the stories from the customer rep who’s also part of their team. The developers provide a high level estimate to develop against each story and, along with the customer, help decide how many of these stories will be implemented during the current iteration. It is worth mentioning that each XP iteration is a release unto itself, not one of multiple iterations toward a single release. Oral communication takes the place of laborious documentation efforts.

XP teams do not spend significant project time on design, either. If an XP pair spends more than 20 minutes working out the design of some supporting classes that would be a lot. XP teams do not design systems for future scalability. They design systems to meet the needs outlined in the stories for the current iteration.

The major work efforts in an XP project center around coding and testing. XP’s paradigm dictates that unit tests are written prior to the targeted code. This is called test-driven development. Refactoring is one of the more interesting concepts that are characteristic of XP projects. The project codebase is owned, collectively, by the project team. Any team member is allowed to optimize any project code as long as no new functionality is added (and all unit tests still pass).

#### 2.3.2 SCRUM

The most distinctive concept in Scrum is the "sprint". Scrum teams deliver application iterations in 30-day sprints. This Sprint can almost be looked at as a mini-waterfall because each classic phase of the lifecycle takes place during this 30-calendar day iteration. Scrum efforts can use existing software development practices when practical. Where XP embraces change, even during an iteration, Scrum assumes that requirements are frozen at the beginning of each sprint. Scrum attempts to achieve business value in each iteration.
Requirements phase-like activities are executed in “Pre-Game Planning” and “Sprint Planning”. In Pre-Game Planning the Scrum team meets with its project stakeholders and collaborates on a list of new features, functional changes, and selected defects that could end up being addressed in this iteration. This list is captured in a Product Backlog. A customer stakeholder is appointed the Product Owner and is the single customer focal point for the project. The Product Backlog goes through a prioritization process until it is scaled down to just what can be addressed in one 30-day sprint. This scaled down and prioritized document is called the Release Backlog. The Release Backlog is frozen for the duration of the Sprint. The Release Backlog is analyzed by the Scrum team and organized into a series of 4-16 hour tasks that would be necessary to complete this Sprint and deliver what’s specified in the Release Backlog. This task list, which is not unlike a project schedule, is the Sprint Backlog. The Sprint Backlog is updated on a daily basis to track progress during each Sprint. Scrum teams assign points to each item on the Product Backlog. The team commits to delivering a certain number of points during each Sprint. After a few Sprints the team understands how much it can deliver and gets more accurate in its estimates.

The Scrum team performs a high level design of items in the Release Backlog. It does not spend significant time on architectural and detailed design. The design evolves as code is written. Some refactoring takes place as the team’s design evolves and clean code is implemented.

2.3.3 Crystal

Crystal is a family of methodologies that are identified by different colors depending on the size and criticality of the project. For instance, Crystal Clear is the lightest weight methodology for small, low criticality projects. At the other end of the spectrum is Crystal Violet that a team would use for large, ultra-critical projects. The differences in each variant of Crystal lie in the degree of rigor applied against each application development activity. Crystal is often thought of as a "stretch-to-fit" methodology. Each Crystal methodology has the same "genetic code". The basic principles of Crystal include:

- **Frequent delivery** (1-3 month customer releases)
- **Close communication** (close team proximity allows rapid feedback that is almost second nature)
- **Reflective improvement** (process improvement from iteration to iteration)
- **Personal safety** (freedom to voice concerns within your project team)
- **Focus** (work on the project with no external interrupts)
- **Easy access to expert business users** (rapid feedback to requirements, design, and interim workproducts)
- **An environment that includes automated testing, configuration management and frequent integration**

One of the more interesting concepts in Crystal is centered on determining how much detail is required for project management, requirements, and design documentation. Documentation detail is determined by the criticality of the project, the risk/severity of undiscovered defects, and the frequency of collaboration available to the Crystal team. Thus, low risk business applications have truly lightweight documentation requirements where high risk defense industry projects (for example) might have documentation requirements reminiscent of a large conventional methodology.

Crystal teams deliver software iterations to the users in the 1-3 month timeframe (aka Frequent Delivery). Crystal advocates frequent integration as well, integrating and building the system as often as is feasible (hourly, daily, weekly).
2.3.4 RUP (Rational Unified Process)

RUP comes from the same iterative and incremental roots as XP, Scrum, and Crystal. RUP, in some ways, is similar to Crystal in that the methodology, rigor, and set of deliverables are recast on a project-by-project basis.

RUP is a methodology that looks quite document heavy but, in reality, advocates a pragmatic approach to all sizes of development projects. RUP utilizes an iterative development approach but, unlike other methodologies discussed in this paper, does not attempt to deliver systems to its end-users with incrementally greater functionality in each iteration. The Unified Process, instead, uses iterations to build functionality that will, ultimately, end up in the final system deliverable.

There are four phases in the RUP lifecycle:

- **Inception** – Establish the business case for the project. Identify all use cases and elaborate the most complex of them. Create a vision document identify the general requirements, features, and constraints. Establish a project plan identifying the phases and iterations. The Inception phase typically lasts no more than several days.

- **Elaboration** – During Elaboration the problem domain is identified and project plan refined. The use case model is further refined to cover 80% or more of the system use cases. A development plan for the rest of the project is created. The team makes a decision on the full set of RUP methodology techniques and deliverables that will be utilized in this project. This is called the “Development Case”. Short and long term architectural decisions are made. The technical underpinnings of the system are iteratively developed. The architecture on which the rest of the system is developed and coded, unit tested, and integrated. This development is best done by small collocated teams.

- **Construction** – During Construction the remainder of the project’s functionality is coded and integrated. If the project is large enough multiple development teams can perform the coding in parallel. It is not necessary to collocate the teams as is recommended in other Agile methodologies. The major product of the Construction phase is a release candidate. The release candidate is an application that’s fit enough to be put in the hands of the end users.

- **Transition** – During Transition the solution is rolled out to the end users for User Acceptance Testing or Beta Testing. After these testing activities the solution may go through a round of parallel testing if it is replacing an existing application. After the production environment is readied, users are trained, and documentation is verified complete the solution is deployed to production or designated for GA (general availability).
3.0 Analysis
There were three major aspects to the analysis performed for this project.

- Identify gaps in the current corporate waterfall SDLC.
- Identify the systems project types that cannot effectively utilize a corporate waterfall SDLC.
- Identify SDLC process features that would be necessary to accommodate these non-waterfall project types.

3.1 Gaps in waterfall SDLC
GE Financial Assurance, Citigroup Asset Management, and JP Morgan Chase have taken a “one size fits all” approach in designing their SDLC methodology. Each is based on a classic waterfall model with a requisite set of documents due in each phase and a requisite set of reviews and approvals associated with each activity. None of these approaches take into account the fact that requirements can be added/changed mid-project. None of these approaches scale the required work and deliverables back depending on project size, budget, and/or risk. Based on the project types I identified in last week’s research work these models will not work well for most heavily customized Package Implementations, most New Development, and will not scale properly for Maintenance.

The Bank of New York has taken a sensible approach in refining their SDLC methodology. They have scaled the phases, deliverables, reviews, and approvals based on the type and size of project. There are categories of projects – Maintenance/Mini, Small, Medium, and Large-scale. These categories are based on financial threshold and effort (in man-months). The Maintenance/Mini project can follow a Fast-path three phase SDLC model. The other categories also scale in approach and there are a number of approved models available that include Waterfall, Incremental, and Iterative depending on the project’s nature and project managers’ choice.

3.2 Non-Waterfall Project Types
The following three project types would best be served by an iterative and incremental SDLC model. Maintenance projects could also be served by an iterative and incremental SDLC in many cases.

- **Package Implementation (COTS)** – Significant customization and configuration of the package is best served by an iterative approach towards delivering the project. It is rare that everything is requirement and integration issue is known up front during a package implementation.

- **New Development (home-grown app)** – Even though New Development projects are intended to follow the initial set of requirements, they rarely avoid mid-stream changes in the project. A waterfall model is not, usually, the best choice for a New Development project of any size.

- **Proof of Concept** – Due to the high degree of configuration and customization required to experiment with new technologies and applications this type of project does not lend itself to the serialized execution required by a waterfall SDLC approach.

3.3 Process Features Required for Identified Project Types
The three project types defined above require an SDLC methodology that allows for iterative and incremental definition, design, and development activities in order to remain in sync with mid-stream changes. In each case, these three phases must be allowed to
iterate until the customer or business approves the implemented feature set as correct for this version of the software/system. This also means that, unlike a corporate waterfall SDLC, documentation deliverables like requirements, architectural design, detailed, design, test plans, configuration management plans, etc. are not frozen before an iteration can begin. All of these artifacts must be controlled through a change management process and versioned. The SDLC model for New Development and Maintenance must be able to deliver frequent incremental and usable value to the end-user. That doesn't mean that every version or revision will make it into the end-user's hands but smaller incremental development will allow the solution to track closer to customer requirements rather than discovering that the solution no longer meets those requirements after a year long project effort.
4.0 Iterative and Incremental SDLC

The following represents the iterative and incremental SDLC model proposed as a result of the Research and Analysis phases of this project.

4.1 Initiate Phase

One of the main determinations made as a result of the Initiate phase is whether a project will be funded or not. In order to make this determination the business stakeholders and their associated IT resources must take the high level project or application concept down to the next level. A Project Charter is created which includes the Product Backlog (entire prioritized feature set), initial project schedule, risks, detailed cost/benefit analysis, resources, Multi-Generation Plan (what will be delivered in each sprint), etc. Another determination that must be made is whether a single Scrum team will iteratively deliver the system/application in consecutive sprints or whether multiple Scrum teams will be formed to execute sprints in parallel. Concurrent Scrum teams would be a necessity for larger systems where multiple subsystems must be developed. In the cases where a single Scrum team is required, the Project Manager would assume the role of Scrum Master. In the case of multiple Scrum teams the Project Manager may assign other Scrum team members (such as a more junior Project Manager or Business Analyst) as the Scrum master for a given team. The Project Manager would assume the role of the Scrum Master for the combined Scrum team (Scrum of Scrums). A tollgate review is scheduled and executed reviewing each detail in the Project Charter. If approved, the Scrum team(s) structure is formed and the project is promoted to the Build Phase. If rejected, and the project is still deemed viable, the necessary changes will be implemented and readied for another tollgate review. If the project is rejected and not deemed viable this project goes no further.
Initiate - Phase

1. Review Project Concepts
2. Develop/Refine Project Charter
3. Document Issue
4. Identify of Issues: Testing: Req'd
5. Identify Make-up Testing: Test: Req'd
6. Identify Make-up Costs: Test: Req'd
7. Develop/Refine WIP: Project Charter
8. Conduct Preliminary Review
9. Preliminary Approval
10. Proceed to Build Phase

A

No

Progress Required

Yes

Final Project Review

A
4.2 **Build Phase**

The Build phase is an iterative cycle of Define, Design, and Develop activities. The initial activity in the Define phase is to firmly establish and refine the Sprint Backlog of requirements that will be addressed within the 30 day Sprint. The Sprint must last 30 days from the start of Define to Deployment. During each iteration there may be a fine tuning of the Sprint Backlog depending on the progress the team is making. In the early stages of implementing the Scrum methodology the Scrum teams may not have a good feel for how many items in a Sprint Backlog can be implemented and rolled out during a Sprint. Once a Scrum team has executed several Sprints it starts to understand its “velocity” and better understands just how many backlog items can be addressed during any given Sprint. During each iteration the Scrum team developers design, code, and unit test at least one Sprint Backlog item (more if there are significant dependencies between items that requires they be developed together). The QA representatives on the Scrum team develop the necessary System Test artifacts (test plan and test cases). A formal Tollgate review will not be held at the end of each iteration just after the final iteration of the Sprint. Daily Scrum team meetings with the customer will replace much of the Tollgate activity. After the coding of each iteration the customer will be involved in some of the integration testing activities to verify the functionality is as desired. After the final Build iteration the customer will accept the feature set for this Sprint is complete as planned. It is now up to the Scrum team to make this “QA” candidate fit for promotion to the Testing phase. Besides making the QA candidate as clean as possible, it is also up to the development team (which may be an external vendor) to supply build, install, and configuration instructions for the next phase.
4.3 **Test Phase (System Test)**

The Test Phase begins with a review of the Build Phase activities with the Scrum Team and its business representative. Code, which has been promoted to the Test Phase, will be rebuilt and compared to the final integration test code. The system will be installed and configured onto the QA environment per documented installation procedures. Previously approved system test procedures and associated test cases, which are part of the Test Plan(s) built during the Build iterations, will be executed against the SUT (system under test). System testing defects will be recorded and quickly reviewed with the Scrum Team members. It is expected that any defect repairs performed against the SUT will be unit tested, re-integrated, regression tested, and the new system version properly labeled prior to promotion to system test. That may translate into additional Build iterations being required to ready the code for UAT (User Acceptance Testing). Once the system test version has been accepted (per previously agreed upon release criteria and completion of all system testing activities), the final System Test code, configuration files, build and installation scripts must be properly labeled and documented prior to promotion to UAT. A System Test Report must be authored by the QA members of the Scrum team detailing all testing activities, test results, defects reported with dispositions, and status against release criteria. A System Test Review will be held with the Scrum Team and its business representative to review the results of this activity.
4.4 Test Phase (User Acceptance Test)

The final system test code version will be rebuilt and compared to last set of executables tested. The Scrum team must provide user level training to the assigned UAT tester(s) prior to the actual UAT. The Scrum team should be available during UAT execution to address potential issues logging defects where appropriate. UAT defects will be recorded and reviewed regularly with the Scrum Team. UAT defect repairs will flow through the Build and Test process with required regression tests and code control prior to being reintroduced in a new UAT code version. When UAT activities have produced an acceptable system version a UAT review will be held to evaluate whether all acceptance test criteria have been met. The final UAT version code will be labeled as a production version. A final Tollgate phase review will be held to review all Test Phase testing activities and determine the promotion to the Deploy phase.

Test Phase – UAT Activity

4.5 Deploy Phase

In this phase the system can be deployed into production but only after some key reviews have taken place. The Scrum Team and business stakeholders conduct a review of the previous phase’s testing activities, verifies that all project deliverables have been updated to reflect the deployable candidate, and that any user documentation and training has been scheduled. A pre-deployment review is held with all responsible support teams to bring them up to speed on the system to be deployed. Any technical details not covered previously by the support teams and Scrum team can be covered in this review. The final review is one dealing with Configuration and Release Management. This formal review, chaired by the Release Management Team, deals with the source code, configuration data, install, and build scripts being readied for deployment and validates that the supporting change control documentation reflects the artifacts being promoted. It also determines whether the appropriate business and technology signoffs have been collected. If one or more configuration items are found not to be in control during the Configuration Management review, Release Management will recommend the appropriate action to be taken. The Core Team has final say and assumes the risk associated with CM defects. A firm deployment date is determined after approval at the Configuration Management review. Once the system is deployed, it goes through a burn-in period which can run from one to four weeks and is determined by the Project Manager and business stakeholders. During this burn-in period all defects that hamper critical functionality are treated as if they are Priority 1 and must be patched immediately. Once the system has successfully gone through burn-in the Project Manager and business stakeholders determine that this Sprint can be closed. A Project Closure Report is created and the project is officially closed. The system version is now in a support mode.
5.0 References


Anonymous, SCRUM – Overview, accessed 2/13/06 from http://www.controlchaos.com/about/


Llopis, Noel, Agile Game Development – Dealing with Chaos In The Real World, accessed on 2/23/06 from http://convexhull.com/articles/agile_development.pdf#search='scrum%20of%20scrums'


