Delta between NAVSEA's ASSET software development life cycle and canonical life cycle models:

A Case Study in Software Maintenance.

Robert W. Schuler
April 2001

Abstract

This paper documents the software development life cycle (SDLC) of the Naval Sea Systems Command (NAVSEA) tool named ASSET (Advanced Surface Ship Evaluation Tool). ASSET is a collection of interrelated naval architecture calculation routines that share a common data base description of a ship. The software development life cycle has been extracted from existing software documentation along with several task statements issued to improve ASSET over the past 25 years.

The SDLC of ASSET is compared and contrasted to several canonical software life cycle models including: waterfall, Mil-Std 498, rapid prototyping, the spiral model, and Donaldson & Siegel's generic four-stage model. Emphasis has been placed on the software maintenance phases of these models.

Introduction

Software Development Life Cycle Models

There are six software development life cycle (SDLC) models discussed in this paper. The first four are well documented in computer science literature, the fifth is defined by Donaldson & Siegel (Donaldson), and the sixth is the actual life cycle of the ASSET program. In the computer science literature, SDLCs similar to the one being followed by ASSET are called "hybrid models" because they incorporate bits and pieces of other models. The well-documented SDLC models are: the waterfall model, Mil-Std-498, the rapid prototype model, and Boehm's Spiral Model. Donaldson & Siegel present an excellent distillation of a four-stage iteration that is at the core of all of the other SDLCs.

In this paper, emphasis is placed on the software maintenance activities in all six SDLC models because this term paper was prepared for Computer Software Engineering course 648 (CSWE 648) "Software Maintenance" during the Spring Semester of 2001 at the University of Maryland University College (UMUC).

Waterfall SDLC Model

There are several variations of the waterfall model available in the computer science literature. The basic idea is that software is developed in a series of stages with each stage providing the input to the next. The diagram used to express this design flow is usually depicted as a series of boxes (representing the SDLC stages) connected by arrows (representing the transitions between the SDLC stages) in such a manner that the diagram resembles a multi-step waterfall. The most sparse waterfall model has only four SDLC stages: define, design, code+test, and deploy+operate+maintain. Davis splits the code+test SDLC stage into two discrete stages and simply calls the last stage "operation".(Davis,8) Rakitin calls the last stage "Maintenance" and further splits the first stage into "Requirements Analysis" and "Requirements Definintion". Figure 1 shows a typical waterfall model.

Davis adds another feature in his representation of the waterfall model. His arrows are bi-directional and reflect the way software is really developed. When errors are discovered downstream (an appropriate metaphor for flowing water), very often adjustments are made to upstream documents and the process is resumed. This does violate the waterfall model's central premise that each stage must be completed before the next one is started, but it is a minor violation.

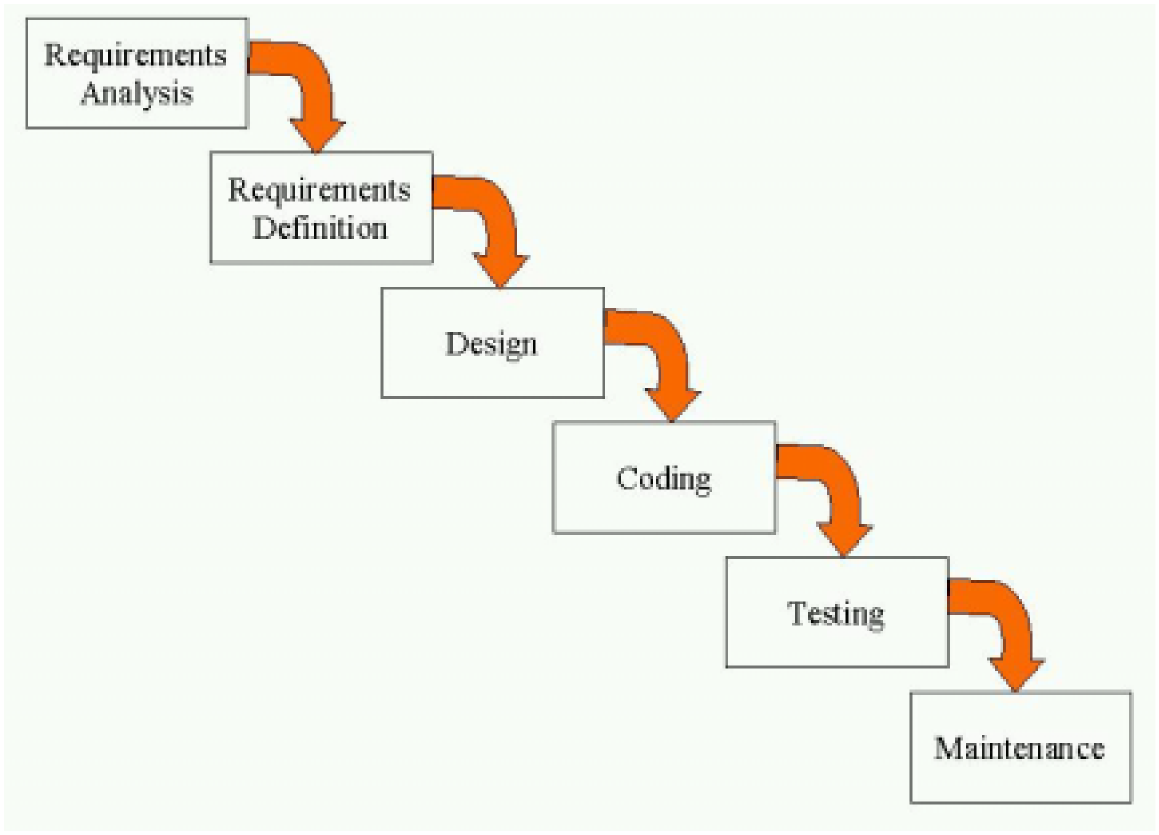

Figure 1 Typical Waterfall Model

Software Maintenance in the Waterfall Model

The waterfall model explicitly highlights maintenance as the stage that follows testing. "Once the client has accepted the product, any changes, whether to remove residual faults or to extend the product in any way, constitute maintenance . . . maintenance may require not just implementation changes but also design and specification changes. In addition, enhancement is triggered by a change in requirements. This, in turn, is implemented via changes in the specification document, design documents, and code . . . it is vital that documentation be maintained as meticulously as the code itself and that the products of each phase be carefully checked before the next phase commences."(Schach, 68)

Mil-Std-498

Mil-Std-498 (December 1994) supercedes DOD-STD-2167A (January 1988), DOD-STD-7935A (October 1988), and (DOD-STD-1703(NS) (12 February 1987). Mil-Std-498 was written to harmonize standards about defense system software development and automated information system documentation. This military standard describes the SDLC as consisting of a series of one or more builds each of which contains one or more computer software configuration items (CSCIs). One of the strengths of this model is that Mil-Std-498 defines 22 data item descriptions (DIDs) which provide excellent guidance in developing documents to support the SDLC.

Each CSCI is developed in accordance with the waterfall model described in the previous section. Mil-Std-498 defines five SDLC stages in the waterfall for a single CSCI: software requirements analysis, software design, software implementation and unit test, unit integration and test, and CSCI quality test. CSCIs may be developed in parallel and Mil-Std-498 defines documentation that must be present before each CSCI is allowed to move to the next SDLC stage in the waterfall.

The computer software configuration item (CSCI) waterfalls are imbedded in an overarching waterfall which consists of six SDLC stages: system requirements analysis, system design, the waterfall for each CSCI, CSCI and hardware integration and test, system quality test, and system use or transition. As with each CSCI, there are documents that must be present before the software is allowed to progress to the next stage of the overarching waterfall.

Software Maintenance in MIL-STD-498

Mil-Std-498 does not directly address software maintenance, however the rigorous configuration management and “software transition plan specification” support the maintenance of a software product. Mil-Std-498 also addresses requirements adjustments and incremental development during a project. However, there is no place within the military standard to feed back bugs from delivered software products or to change the original requirements that kicked-off the project. These two activities (bug reporting, and requirements changes) take place outside of the scope of the military standard. Software that is developed by following the rigor of the standard, however, will be easy to maintain.

Rapid Prototype SDLC Model

The basic idea of the rapid prototype SDLC is to get software into the hands of the end user quickly for the purpose of more clearly understanding and defining the customer's needs and the software requirements. In a way, this SDLC uses a computer language (like C or FORTRAN) to "document" the customer's requirements. Unlike the waterfall model, the rapid prototype model is based on a short cycle iteration that consists of four primary SDLC stages: gather requirements, perform a quick design, build a prototype, customer evaluation of prototype.(Rakitin,18). Based on the customer's response, the cycle is repeated resulting in a refined prototype. Eventually, the developer and the customer reach a point where more disciplined software engineering may begin.

A big risk to following the rapid prototype SDLC is that "frequently, the prototype is not thrown away and becomes the product. Depending on how the prototype was developed, this can result in major problems for long-term support and maintenance of the product."(Rakitin, 19). To avoid this pitfall, some organizations call this the "throwaway approach" to prototyping.(Davis,341). Other organization plan to have the prototype evolve into the system and call this the "evolutionary approach" to prototyping. (Davis, 341)

Software Maintenance in the Rapid Prototype Model

As with MIL-STD-498, maintenance is really outside of the scope of the rapid prototype model. The changes suggested by users during the iterations of the rapid prototype are really part of the requirements gathering activity. In the case where the development plan calls for the prototype to evolve into the final product, there is very blurry distinction between when user feedback stops being requirements refinement and actually becomes part of a maintenance activity. At some point in time, however, the software development effort runs out of funding or hits a calendar deadline. After this point, all requests to change the software product fall under the umbrella of software maintenance.

Boehm's Spiral SDLC Model

A formal attempt of combining the benefits of rapid prototyping with more formal SDLC models is the spiral model developed by Boehm.(Rakitin, 19). This model combines the "evolutionary approach" to prototyping with a four-step cycle that provides metrics for each iteration of the software design. The four steps are: planning, risk analysis, development, and assessment. The spiral model is a risk driven model as opposed to a document driven model (like Mil-Std-498 or the waterfall model) or a code-driven model (like the rapid prototype). (Boehm, 1988, p.61) The spiral is shown on a Cartesian plane with each quadrant representing one of the four steps. The radial distance from the origin to the spiral represents the cost of the project and the distance traveled along the spiral represents time. Each loop on the graph represents an evolutionary protocol. Since each loop of the spiral surrounds the previous loop, this model shows that each prototype takes a little longer than its predecessor and adds cost to the project at a slightly faster pace. Figure 2 shows Boehm's original spiral model.

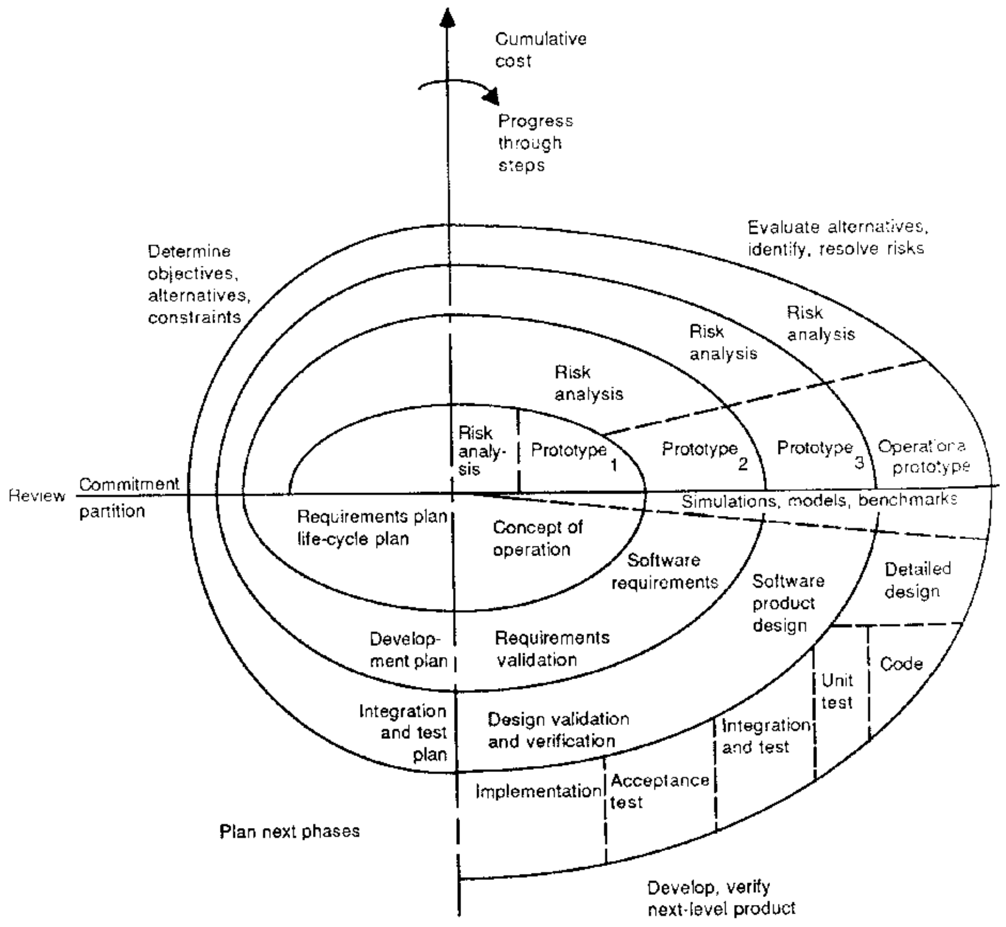

Figure 2. Spiral model of the software process.

Copied from IEEE Computer Magazine, Volume 21, May 1988 Page 64
Boehm, B. "A Spiral Model of Software Development and Enhancement"

Figure 2 Boehm's Original Spiral Model

"During the second trip around the spiral, a more refined prototype is built, requirements are documented and validated, and customers are involved in assessing the new prototype. By the time the third trip around begins, risks are known, and a somewhat more traditional development approach is taken." (Rakitin, 22)

Since the publication of the spiral model in 1988, the model has been expanded to take into account Theory W, a management theory based on ensuring project success by making winners of the system's stake holders. This revised spiral model is presented in "Using the WinWin Spiral Model: A Case Study" (Boehm, 1998, July). Figure 3 shows Boehm's updated spiral model.

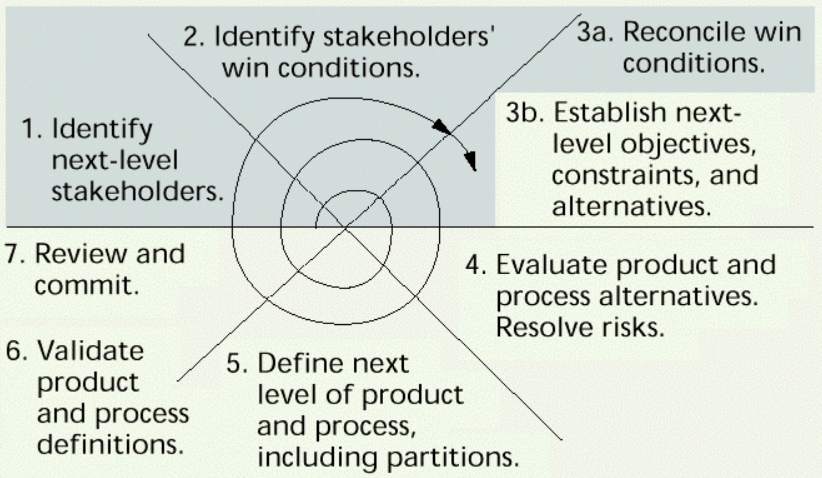
2. Identify stakeholders' win conditions.
3a. Reconcile win conditions.
1. Identify next-level stakeholders.
3b. Establish next-level objectives, constraints, and alternatives.
7. Review and commit.
4. Evaluate product and process alternatives. Resolve risks.
6. Validate product and process definitions.
5. Define next level of product and process, including partitions.

Figure A. How the WinWin spiral model differs from the original spiral model. The new model adds front-end activities (blue) that show where objectives, constraints, and alternatives come from. This lets users more clearly identify the rationale involved in negotiating win conditions for the product.

This figure has been copied from "Using the WinWin Spiral Model: A Case Study" B. W. Boehm et al (Computer, July 1998 p 34)

Figure 3 Updated Spiral Model

Software Maintenance in Boehm's Spiral Model

As with MIL-STD-498 and the Rapid Prototype Model, maintenance is really outside of the scope of the Spiral Model. The changes suggested by users during the orbits of the spiral are really part of the requirements gathering activity. Just like with the rapid prototype model, as the software evolves into the final product, there is very blurry distinction between when user feedback stops being requirements refinement and actually becomes part of a maintenance activity. At some point in time, however, the risk level is sufficiently reduced and a more traditional SDLC model is adapted. After the point in the traditional SDLC when the product is delivered to the customer, all requests to change the software fall under the umbrella of software maintenance.

Donaldson & Siegel's Generic Four-Stage Model

Donaldson & Siegel present the SDLC as an object with four "interfaces", as shown in Figure 4.(Donaldson, 14-16) The first interface is geared towards answering the question: "What is the system to do?". The second interface addresses the question: "How is the System to do it?". The third interface interacts with the process of building the software, and the fourth interface interacts with the process of running the software. By invoking the interfaces in the order just listed (What, How, Build, Use), we have a process similar to the canonical SDLCs described above: Waterfall, MIL-STD-498, Rapid Prototype, and Spiral.

At the center of the four-stage model is a Review Area comprised of Product Assurance and Management "methods". Each change of requirements (What) is fed into the object and processed by the management method. Each change in architecture (How) is also fed into the object and processed. Building schedules and budgets are also controlled by the object, as are bugs and enhancements discovered during the use of the resulting software. By treating the Generic Four-Stage Model as an event driven object, we can gain a unique and useful perspective about the SDLC and software maintenance.

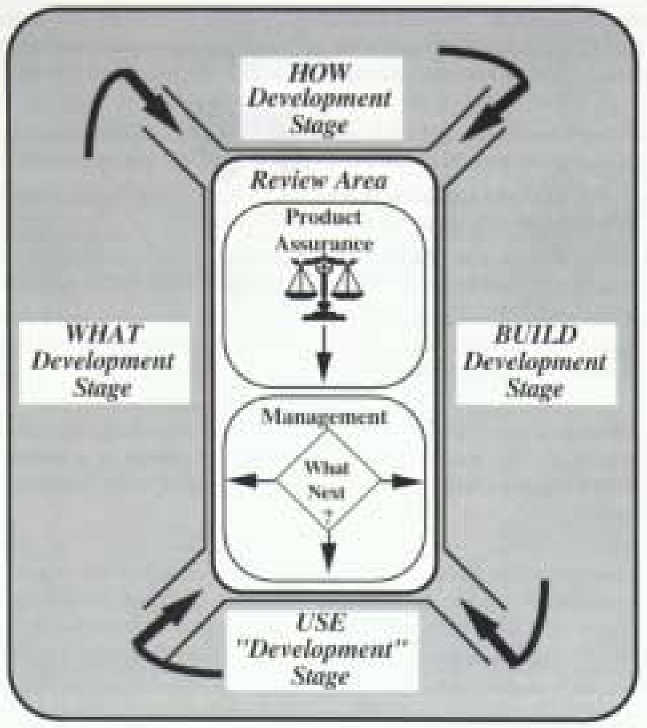

FIGURE 1–8 Our four-stage generic life cycle blends the requisite software systems development disciplines of development, product assurance, and management.

This figure has been copied from Cultivating Successful Software Development , A Practitioner's View by Donaldson, S. E. & Siegel, S. G. ©1997 by Prentice Hall PTR; ISBN 0-13-341678-X

Figure 4 Donaldson & Siegel's Generic Four-Stage Model

Software Maintenance in Donaldson & Siegel's Generic Four-Stage Model

Since all four interfaces of the Generic Four-Stage Model are essentially “event driven”, they all impact software maintenance. If the user requirements change, the necessary updates to the software are described by the Review inside the object. This is defined as revolutionary or unplanned change. (Donaldson,128) If the architecture is altered to adapt to a new technology, then the "How" interface triggers the Review which updates the appropriate design and documentation information. Likewise, information through the "Build" interface impacts schedules and budget considerations. Finally metrics from software "Use" can be used to trigger changes to requirements, or architecture. These changes are also revolutionary. "Use" also discovers and reports bugs, which must then be corrected. Correcting bugs is generally considered evolutionary or planned change, unless the bug is due to an error in specification or architecture. (Donaldson,128)

ASSET's SDLC

A full narrative description of ASSET's software development life cycle is provided in the next section of this paper. This section attempts to provide a sense for how to interpret the following 'case study' in terms of the textbook SDLC models outlined so far in the paper.

Tracking the history of a large software project over the past two and a half decades has not been easy. The primary government agency involved has been renamed several times. Current programmers are second and third generation from the perspective of the project, and there is no one left from the original effort to answer questions.

Fortunately, a very good set of documentation was generated in 1990 when (it appears) Boeing turned over ASSET to NSWC-CD. Unfortunately, this documentation is more focused on the details of operating the software than on the management details of the software project. In addition to the software's documentation, I have also been able to review a few Engineering note books that contain contract numbers, dollar amounts, and a rough sense of how the ASSET project progressed.

From its inception, ASSET has been an Engineering tool (actually a Naval Architect's tool) and the programmers who have been involved have primarily been oriented towards the rapid prototype SDLC model. By and large, ASSET has been developed one module at a time, where each module has been prototyped, compiled, tested, and released to end-users for validation.

The user community of ASSET is currently under 100 individuals, with less than 20 power users world wide. This makes the incorporation of end users into the software life cycle both desirable and cost effective. As will be illustrated in the following case study, users state new requirements and users validate that the software meets theses requirements.

Software Maintenance in ASSET

ASSET's software requirements are not kept in a formal document, but rather in the collective awareness of the Naval Architecture community within the Navy. The only performance requirements seem to be that ASSET must produce results consistent with current Naval Architecture practice. Test cases exist in the form of ship models of real ships. When ASSET's results deviate too much from the known characteristics of real ships then it is known that some part of the program is invalid.

User requests for changes, are tracked in a “living document” that is maintained by the Development Team Leader. There are actually three documents maintained to track requests for change: a long-term wish list, a short-term project specific list, and a release-plan list. User’s feed requests for change into the system via e-mail, a web-based form (http://www.dt.navy.mil/asset/TechSupp/techform.htm), and verbal requests directly to the ASSET development staff. Programmers also point out necessary changes (long term and immediate) that are uncovered as a result of fixing bugs or enhancing the tool set.

There are four primary kinds of changes identified by the Development Team Leader: extensions to the tool set (adding new capabilities to ASSET), enhancements (extending an existing capability—usually this is to add a higher level of granularity to the data model and calculations), bug fixes, and changes to the user interface. The fact that changes to the user interface are treated as a separate category distinct from extensions and enhancements reflects the culture that uses and develops ASSET. Higher priority is given to the calculation facilities of ASSET than to the user interface.

The ASSET Lifecycle

The Advanced Surface Ship Evaluation Tool (ASSET) is a family of interactive computer programs for use in the exploratory and feasibility design phases of Navy surface ships. ASSET is made of several computational modules each of which addresses a specific domain of naval architecture, such as hull geometry, hull structure, resistance, propulsion, machinery, weight, space, hydrostatics, seakeeping, manning, or cost. These modules provide both design synthesis and analysis capability. These modules are configured into several executable programs, each one customized to the characteristics of a specific type of hull, called a ship-type. To provide ease of use of the entire ASSET system, each ship-type program shares a common Executive Program, which provides not only a friendly user-computer interface, but also provides a unique data management system which simplifies the storage, recall, and use of ship data by an ASSET user.

NOTE: Most of this description of ASSET and its history comes from the ASSET System Manual, Volume I. (Devine, 1991, pp1-5).

HANDE

The origin of ASSET is a program that was known as the U.S. Navy Hydrofoil Analysis and Design (HANDE) computer program, as described by Devine and King (Devine, 1981, January). HANDE was built in the mid 1970's to provide exploratory and feasibility design support for hydrofoil ships. HANDE eventually to became one of the family of ship-type programs within the ASSET framework.

The HANDE computer program development formally began in December 1972 with an award to The Boeing Company of a HANDE Phase 0 development contract by the Hydrofoil Office of the David W. Taylor Naval Ship Research and Development Center (DTNSRDC). The contract award followed several Boeing proposals to the Navy for the development of the HANDE computer-aided engineering concept. Under Phase 0, the technical content and the overall system design were to be clearly and explicitly defined. HANDE Phase 0 successfully concluded with the publication of a software design document by Brennan, Burroughs, Hurt, Wichert, and Wacker (Brennan, 1973).

Construction of the HANDE computer software was initiated following the close of Phase 0. The bulk of the software development occurred during HANDE Phase 1, which commenced in March 1975. Several preliminary program versions were made operational on the Boeing EKS computer installation before the first fully operational HANDE program, Version 0.0, was assembled and delivered to the Navy in April 1977. Nearly 30,000 man-hours of management, engineering, programming, and clerical labor were spent to make this initial version of HANDE operational.

Since its initial installation, several new HANDE versions (numbers 1.0, 1.1, 1.2, 1.3, and 1.4) were developed that improved program performance and expanded the technical capability of the program. Each subsequent version of HANDE replaced the previous version on the DTNSRDC computer system.

HANDE SDLC Analysis

It can be seen in this discussion of HANDE that the initial software development took place in two phases, and that the first phase was clearly a requirements gathering and software design effort. No actual coding took place until the second phase. This is similar to the opening stages of the waterfall model. Furthermore, the series of HANDE versions can be interpreted as the maintenance stage of the waterfall model. It is also possible to argue that the HANDE model transitioned from a 'pure' waterfall life cycle to the spiral model since each subsequent version "expanded the technical capability of the program." By planning the initial development in two phases, HANDE was structured to follow the spiral of expanding capabilities over time. In other words, the maintenance and upkeep of the tool evolved directly from its initial SDLC model.

Early ASSET

In 1980, after an examination of existing ship synthesis tools, the Technical Director's Technology Application Team (TDTAT) #1 at DTNSRDC embarked on the development of ASSET, which was to be a computer program patterned after HANDE that would be applicable to the feasibility and concept design of conventional frigates, destroyers, and cruisers. The initial step in development of ASSET was the creation of Version XX, which was to be a limited tool whose primary purpose was to demonstrate ASSET program concepts, and to provide an initial framework for future program development. ASSET Version XX was derived from the HANDE computer code by removal from HANDE of all algorithms and parameters pertaining to foil/strut/pod systems. This work was performed by The Boeing Company under contract to DTNSRDC and was concluded in January 1981 with the publication of the ASSET XX User Guide (Devine, 1981, February).

Following a brief period of testing and evaluation of ASSET Version XX, work was started on the first fully operational release of ASSET, Version 0. Development of the ship-design technology incorporated in Version 0 was jointly performed by the DTNSRDC Technical Director's Technology Application Team #1

(DTNSRDC Code 014), the Technical Director's Technology Application Team #2, the DTNSRDC Systems Development Department (Code 117), and The Boeing Company. The Boeing Company, again under contract to DTNSRDC, was assigned the task of developing the ASSET Version 0 computer code and of developing a complete set of supporting documentation. The ASSET Version 0 development effort successfully concluded in May 1982 with the installation of the program on the DTNSRDC computer installation and with the publication of a complete set of supporting documentation.

Early ASSET SDLC Analysis

Version XX of ASSET is more of a rapid prototype since its primary purpose was to demonstrate the feasibility of future software development. In this case, the customer was already aware of the predecessor software HANDE. ASSET XX was an evolutionary prototype rather than a throw away prototype since its initial framework was carried on to version 0. This corresponds to the customer evaluation stage of the rapid prototype model. It is a valid interpretation to say that ASSET grew out of the maintenance phase of HANDE.

ASSET Version 1

Following the installation of ASSET Version 0, work began on a new release of ASSET, called ASSET Version 1. The major enhancement to ASSET for this new version was the incorporation of many of the algorithms and methods utilized by the Naval Sea Systems Command computer program known as the Destroyer Ship Synthesis Model, or DD08. Other improvements to ASSET were also made for Version 1. Version 1 was made fully operational on the DTNSRDC system in June 1983. A revised set of documentation was also published at that time.

After the delivery of Version 1, efforts were initiated at DTNSRDC to develop a capability similar to that offered by HANDE and ASSET for small waterplane area twin hull (SWATH) ships. The completion of theoretical development by Edkins and Mulligan (Mulligan, 1983) enabled code development to begin in late 1983. An initial prototype capability for this program, then known as SWATHET (SWATH Evaluation Tool), was brought online in April 1984.

Concurrent with the development of SWATHET, the Advanced Concepts Office (Code 117) at DTNSRDC began an effort to assemble existing Navy synthesis tools into a single, unified framework. The name of this unified framework was borrowed from the ASSET program. Whereas the ASSET name had previously been associated with a single program, it now was associated with a family of programs, each having the same general architecture and the same general capabilities. Each program within the ASSET umbrella was to be used with respect to a specific type of Navy ship, such as monohull surface combatants, SWATH ships, hydrofoil ships, planing craft, SES ships, ACV's, etc.

As of December 1984, three types of ships were represented within the ASSET umbrella of programs, as described by Clark, Fein, Jones, and Sheridan (Clark, 1984). The first program was the monohull surface combatant program (ASSET/MONOSC), which was the ASSET Version 1.1 program that became operational in 1983. The second program was the SWATH ship program (ASSET/SWATH), which was formerly known as SWATHET. The third program was the hydrofoil ship program (ASSET/HYDROFOIL), which was formerly known as HANDE. Although each ASSET program has technological capabilities that are unique to the ship type, the programs all share a common executive, and many of the programs also share some common technological capabilities that are independent of ship type.

ASSET Version 1 SDLC Analysis

At this point, the SDLC deviates from all four of the canonical SDLC models because it merges technology from several projects into a common framework, while simultaneously advancing the individual subprograms under a common umbrella. This sort of harmonization of disparate but similar software development efforts does not exist in the cannon of SDLC models. From the perspective of the ASSET project, version 1 considers all previous efforts as prototypes to be reworked into a new common framework.

Donaldson & Siegel's generic four-stage model does allow this migration path to be modeled. The "HOW" portion of the model is unchanged from a technology and framework perspective. The "WHAT" portion of the model is expanded to a superset of all of the subordinate projects. The changes are fed into the appropriate interfaces in the model, and a new project schedule is produced by the management method in the middle of the model.

The development of SWATHET reflects the simplified waterfall approach used throughout ASSET. Specifically, there are two steps in this simplified waterfall: 1) develop the theory, the 2) write the code. Testing and validation are accounted for when the end-user is satisfied that test cases properly reflect the theory.

The development of SWATH software also highlights another key element which distracts from the continuity of management required when following canonical SDLC models: project funding and therefore the customer changes based on the annual USN budget priorities. Again, this is accounted for in the "management" part of the Donaldson & Siegel's model.

ASSET Versions 2 and 3

Concurrent with the implementation of the ASSET umbrella in May 1984, plans were finalized to merge various Naval Sea Systems Command (NAVSEA) programs, and particularly the Destroyer Synthesis Model (DD08), into the ASSET/MONOSC program. This merger was completed in 1986 with the release of ASSET version 2.

After version 2, ASSET was expanded to address additional types of surface ships of interest to the U.S. Navy. Under the general heading of ASSET, ship-design synthesis capability similar to that which exists for monohull surface combatants, SWATH ships, and hydrofoil ships is being developed or was planned for the following ship types: small ocean-going vessels, amphibious ships, air cushioned vehicles, planing craft, underway replenishment ships, monohull aircraft carriers, and surface effect ships.

Asset version 3 was released in the early 1990s. This version of the software had several improvements in the ship synthesis algorithm used to balance several competing ships characteristics like weight, stability, and range. Another major feature of version 3 was the addition of a module for laying out and accounting for major ship machinery like propulsion plants.

ASSET Versions 2 and 3 SDLC Analysis

This represents a shift in the level of detail accounted for in the ASSET data model. While the basic function of the software was a refinement of previous versions, this new granularity greatly enhanced the software's utility. These versions follow Boehm's Spiral SDLC model. Since the level of risk for introducing new capabilities was reduced by the stable software framework, it was easier for the project to add new capabilities and to support new ship-types. This part of ASSET's history illustrates the statement that "good software gets maintained".

ASSET Version 4 to Present

In 1995, ASSET version 4 was released. The most significant change in version 4 was that all program code was ported to run under Microsoft Windows NT or Windows 95 to take advantage of 32-bit architecture. Version 4 also introduced a new ship type program for aircraft carriers, MONOCV. The primary ship-type programs under ASSET version 4 where: MONOCV, MONOLA, and MONOSC.

The capability to export data to other programs was significantly enhanced. In previous versions, several modules might have to be run by the user to generate the desired output. In ASSET Version 4, an Export Utility was introduced to automate and expand this capability and included an export to NAVSEA's CAD2 platform to allow the generation of a 3-dimensional model of the ship in the Intergraph Vehicle Design System (VDS).

ASSET Version 4 added the capability to make a direct link over a network to a workstation running the Parametric FAST SHIP hull form modeler. FAST SHIP is the primary hull form modeler used by NAVSEA 03H. The "Parametric" part of FAST SHIP is a series of macros that control the modification of hull principal dimensions and coefficients by FAST SHIP. The FAST SHIP macros follow a method similar to ASSET's Hull Geometry Module.

There where several incremental releases of ASSET Version 4 during 1996: 4.01 on September 4, 1996; 4.02 on Octoboer 4, 1996; 4.03 on October 31, 1996; and 4.04 on November 22, 1996. These all corrected minor bugs and expanded existing capabilities. These where then followed by version 4.1 July 1997, 4.2 in October 1997, 4.3 in September 1998, 4.4 in February 1999, 4.5 in April 2000, and 4.6 in October 2000.

For the most part these point releases corrected bugs discovered by users and added additional functionality requested by users. One feature that bears an extra note is the machinery wizard that was added to version 4.2. This interactive tool assists the user in making several design decisions in populating ASSETS ship parameter data. This represents a step toward expert system development, because the machinery wizard captures the design process followed by senior Mechanical Engineers and uses friendly windows prompts to assist junior engineers and Naval Architects to make sensible decisions.

Current efforts in ASSET development are focused on revising the underlying data model used to store and represent details about the ship. While it is possible that there will be several interim releases of the software, once the new data model is incorporated into the program, version 5 will be released.

ASSET Version 4 to Present SDLC Analysis

ASSET is currently evolving following a hybrid approach SDLC model. New modules are developed following a rapid prototype approach. The customer for the product is not always clear and seems to change from fiscal year to fiscal year adjusting to new Naval ship acquisition projects as appropriate. The steady stream of point releases is similar to the software builds described in MIL-STD-498. The difference being that ASSET is not following the strict documentation requirements of the military standard.

Modules and updates are made in a short code/compile/test cycle, and all source code is kept under revision control. When end users get unexpected results or intuitively challenge ASSET's calculations, they call and discuss the discrepancies with the ASSET programmers. The Naval Architecture theory and Numerical Method theory upon which ASSET is based are available to end users and the user community is largely composed of engineers.

From a macro perspective, ASSET is in the maintenance mode of the canonical waterfall model. The basic function of the program (to help synthesize early stage ship designs) is not changing—rather it is just being refined. At a micro level, however, each change to the underlying data structure or granularity of design follows its own SDLC (usually a modified spiral model).

Real World Vs Textbook

As is typical of Engineering textbooks, computer science textbooks are full of simplified models that are suitable for illustrating key concepts in a classroom, but very seldom translate directly into real world situations. All five of the software development life cycle (SDLC) models presented at the beginning of this paper are introduced under certain assumptions when they are presented in textbooks.

A fundamental assumption is that management will have continuous and consistent control of the software product throughout its entire lifecycle. This assumption of a clean slate from which to start greatly simplifies the presentation of two important programming concepts: 1) define before design, and 2) design before coding. As was illustrated in the above case study of ASSET, in the real world software engineers are often asked to start from existing code. It is still possible to define and plan using the existing code as a constraint, but this greatly complicates textbook models. For example, if there is existing code that must be incorporated into new work, how should requirements and their traceability to the existing code be handled?

Are project managers allowed to have "because it was like this when we got here" as a design criteria?

Another common assumption is that only one platform or technology will be used throughout the life cycle of the software. Boehm addresses this concern by making the assessment of platform change one of the risks to be assessed before each trip around the spiral. Mil-Std-498 addresses this concern by requiring the hardware to be specified as well as the software. Donaldson & Siegel address this by separating the “WHAT” from the “HOW”. Many of the point releases of ASSET from version 4.0 until the present where due to bugs introduced by porting the program into the Microsoft Windows environment.

Several of the models recognize that the customer requirements will change, but none of the models elaborate this to the point where the actual customer changes. For example, ASSET started out trying to satisfy the specific needs for hydrofoil designers, was next asked to adapt to single hulled surface combatants, then small waterplane area twin hull ships, and finally air craft carrier designers. Not only did these changes affect the software requirements, but they also changed project planning and financing. Of the five models presented at the beginning of this paper, Boehm's spiral model comes the closest to matching this aspect of ASSET's evolution. By assessing the risk of customer changes and by properly planning the next trip around the spiral, managers following the spiral model can adjust to a seemingly schizophrenic customer. Furthermore by shifting from the 1988 version of Boehm's spiral model to the 1998 version that also accounts for stakeholder motivations in addition to project risks, management might find a tool that can aid in ASSET's evolution.

It is up to managers to transition the ideas presented in the textbooks into real world situations. The simplifying assumptions used to present models in textbooks do not invalidate the principles being illustrated. Rather, these assumptions help distill important ideas for easy consumption.

Conclusions and Recommendations

The subtitle of this paper is "a case study in software maintenance". ASSET is not currently following a formal software development life cycle (SDLC), however ASSET continues to improve every year. More importantly, the customers are satisfied with ASSET's performance. The primary impact of the informal approach seems to be a lack of measurable feedback that could be used to gage the maturity of the ASSET development team and the effectiveness of management decisions.

It is recommended that ASSET managers review the formal SDLC models presented in this paper to determine which techniques and principles may be applied to the current development effort. This paper is intended to form the initial step towards meeting this need.

It is strongly recommended that ASSET managers adopt a more formal approach to capturing user requirements and measuring the ASSET developer’s effectiveness in meeting those requirements. This will provide metrics useful in developing long-term strategies and may help to break the pattern of adjusting course every time Congress (and the Navy) issue a new budget.

Finally, it is recommended that the ASSET documentation be updated to reflect the current state of the software. Some of the documents referenced in writing this paper have not changed since 1990! Formal documentation of the test data used for internal verification along with a more formal mechanism for recording user validation efforts would greatly assist newcomers in assessing the quality and capability of the Advanced Surface Ship Evaluation Tool.

References

Boehm, B. (1988). "A Spiral Model for Software Development and Enhancement" (IEEE Computer, Vol.21, May 1988, pp 61-72)

Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.,(1998) "Using the WinWin Spiral Model: A Case Study" (IEEE Computer, July 1998, pp 33-44).

Brennan, A. J., Burroghs, J. D., Hurt, W. C., Wichert, W., Wacker, D., (1973) "Hydrofoil Analysis and Design Program (HANDE) Final Report". The Boeing Company, Seattle, Washington, D221-51302-1, June 1973.

Clark, D. J., Fein, J., Jones, R., Sheridan, D.,(1984) "The ASSET Program - A Current Navy Initiative", Society of Naval Architects and Marine Engineers STAR Symposium, April 1984.

Davis, A. (1993). Software Requirements Objects, Functions & States. Prentice Hall PTR. ISBN 0-13-805763-X

Devine, M. D., (1990) "Advanced Surface Ship Evaluation Tool (ASSET) System User Manual". The Boeing Company, Seattle, Washington, BCS 40529-0, May 1990.

Devine, M. D., (1981) "Advanced Surface Ship Evaluation Tool (ASSET) User Guide". The Boeing Company, Seattle, Washington, BCS 40329, January 1981.

Devine, M. D., King, J. H.,(1981) "HANDE - A Computer-Aided Design Approach for Hydrofoil Ships," Naval Engineers Journal (ASNE), April 1981.

Donaldson, S. E., Siegel, S. G.,(1997) Cultivating Successful Software Development, A Practitioner's View. Prentice Hall PTR, ISBN 0-13-341678-X

MIL-STD-498 was available on-line April 30, 2001 at: http://www2.umassd.edu/SWPI/DOD/MIL-STD-498/ROADMAP.PDF

Mulligan, R. D., Edkins, J. N.,(1983) "Small Waterplane Area Twin Hull Advanced Surface Ship Evaluation Tool - Preliminary Theory Manual", David W. Taylor Naval Ship Research and Development Center, Advanced Concepts Office, April 1983.

Rakitin, S. R., (1979). Software Verification and Validation A Practitioner's Guide. Artech House. ISBN 0-89006-889-5

Schach, S. R.,(1999). Classical and Object-Oriented Software Engineering with UML and C++. WCB/McGraw-Hill. ISBN 0-07-290168-3