CHUNK: An Agile Approach to the Software Development Lifecycle

Andrew Aken

ABSTRACT: This paper introduces a methodology for developing applications incorporating components of Agile development methods with the traditional Software Development Lifecycle (SDLC), a/k/a the "waterfall" model of software development. We look at the causes for failure of software development projects and propose steps in the development process to address many of the root causes of these failures (it would be implausible to state that any development methodology could eliminate the risk of failure). We also provide a critical review of Agile principles and the traditional SDLC. The steps of the new methodology are then described with rationalization for their necessity.

KEYWORDS: Agile methodologies, design, requirements analysis, software development life cycle (SDLC), software development methodology, system analysis

---

Andrew Aken is a Lecturer in Information Systems and Applied Technologies, College of Applied Sciences and Arts, Southern Illinois University, Carbondale, IL 62901.

Address correspondence to Andrew Aken, Lecturer, Information Systems and Applied Technologies, College of Applied Sciences and Arts, Southern Illinois University, Carbondale, IL 62901. E-mail: ajaken@cba.siu.edu.

Journal of Internet Commerce, Vol. 7(3) 2008
Available online at http://www.haworthpress.com
© 2008 by The Haworth Press. All rights reserved.
doi: 10.1080/15332850802250385

Web-based application development is not so fundamentally different from traditional software development that it can wholly disregard utilization of development methodologies. In fact, it can be argued that given the tight integration of many web-based applications with the rest of the organization (e.g. e-commerce applications must work within the bounds of the marketing department, customer service, accounting, supply-chain management, etc.), it can be even more critical to utilize a comprehensive software development methodology (Boehm & Basili, 2001; Glass, 2003).

As in the early years in software development, web-based application development is often accomplished without any formal methodology applied. Many early web-based projects were developed by novice developers without any formal training in, or understanding of, software development and utilized an Ad Hoc approach. Many web design organizations still function in this manner. Even when a software development methodology was utilized, the stakeholders in the project are frequently not properly identified. Stakeholders are generally recognized as the visitors to the web site, but may also include innumerable individuals and departments within the organization as well.

However, several concerns have also been expressed for many years regarding traditional software development methodologies. Baskerville and Pries-Heje (2004) identified 6 significant problems with the way most organizations approach software development:

- Formal methodologies within an organization are often poorly documented or poorly applied.
- Identification of key stakeholders is typically incomplete.
- Requirements analysis is not performed or is deficient.
- Requirements are changed frequently throughout the system development.
- Problems with requirements, design, or implementation aren't discovered until the product is delivered.
- Inability to deliver applications with "Internet Speed".

To rectify many of these deficiencies, "Agile" software development methodologies were introduced in the late 1990s and early 2000s. As stated in the Agile Manifesto (Agile Alliance, 2001), the Agile Alliance values:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

However, because of their apparent lack of rigor in applying a systematic requirements analysis and the fundamental shift in the process of software development proposed by Agile development methodologies, they have been erroneously categorized as being amethodological. Agile methodologies, though, have well-defined processes for stakeholder interaction and development.

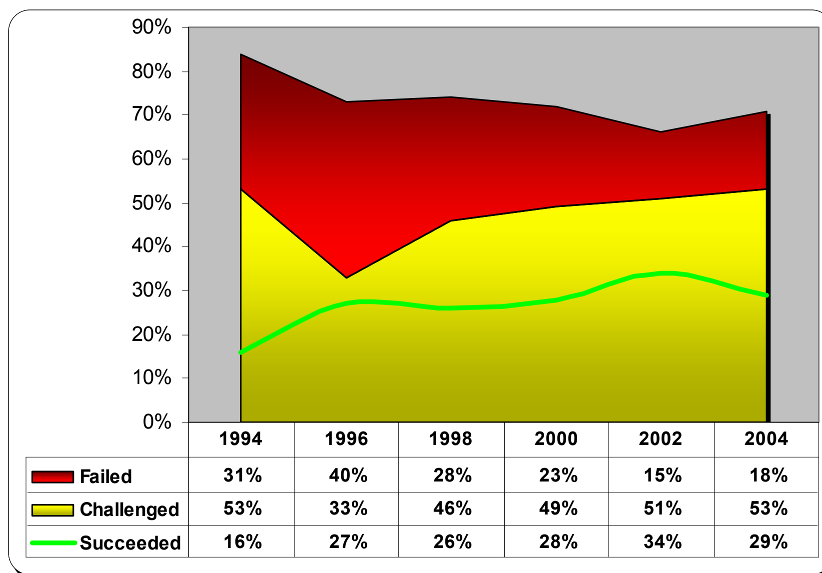
This paper will look at the current state of software development to determine if improvement is necessary. The paper will then look at many of the causes of software development failures and determine if the available methodological choices address these causes. The paper will then propose a software development methodology which attempts to address the primary flaws leading to software development failures by taking advantage of the benefits of both Agile methodologies and the traditional SDLC.

SOFTWARE DEVELOPMENT FAILURES

Although application development failures are in a general decline since the Standish Group first published it's rather alarming CHAOS Report (Glass, 2006), the number of failed (cancelled or finished but not used) and challenged (over budget, exceeding estimated time, or lacking features) projects still comprises 71% of projects in surveyed organizations (Hartman, 2006).

Although the methods used in collecting the data for the Standish report have come into question (Glass, 2006), the trends and basic findings of the studies are reflected in other research as well (Ewusi-Mensah, 2003; The Hackett Group, 2003).

FIGURE 1. Software Implementation Success/Failure (Hartman, 2006)



	1994	1996	1998	2000	2002	2004
Failed	31%	40%	28%	23%	15%	18%
Challenged	53%	33%	46%	49%	51%	53%
Succeeded	16%	27%	26%	28%	34%	29%

In Ewusi-Mensah's (2003) book, Software Development Failures, he analyzes the reasons for project abandonment at 54 organizations. His conclusions state that current software development methodologies have not succeeded in providing reliable, valid, and verifiable ways to combat software project failures. Failure or abandonment of a project may occur whenever the expectations of any of the stakeholder groups become unrealized and the situation causes management to terminate the project prior to its installation and operation.

Another outgrowth of poor methodological choice and/or utilization is the significant increase in the costs of software maintenance. As much as 80-90% of the total costs of software development are incurred during maintenance (Bennett, 1991).

METHODOLOGICAL RESPONSES TO SOFTWARE IMPLEMENTATION FAILURES

Many of the causes of software implementation failures and maintenance costs can be attributed to the methodology (or lack of methodology) used or its execution. In the following sections, we will

look at several of the aspects of software development which have been shown to be the primary contributors to implementation failures and how these are addressed by the SDLC and Agile methodologies. We will also look at how these attributes should be addressed in a software development methodology to increase the likelihood of a successful software development project.

Software Development Iteration

One of the primary reasons for these high failure rates is the delay between the specification of the functional requirements and the final installation of the working system. This delay causes many potential problems that remain hidden throughout the duration of the development process (MacCormack, 2001; Boehm, 2002; Ewusi-Mensah, 2003; Poole, 2006; Nerur & Balijepally, 2007). To combat this problem, the authors have recommended a need for iteration and recursion in the development of the software project. The intention is to bring functional code to the customer earlier in the process so that mistakes in the requirements analysis can be brought to light sooner in the development cycle.

One of the primary benefits of Agile methodologies is that they iteratively produce deliverable products. This allows the developers to quickly generate value and receive feedback faster. The quicker the feedback is received from the customer, the earlier problems can be discovered and resolved (MacCormack, 2001; Poole, 2006).

The Agile Manifesto also stresses the importance of delivering working software frequently (from a couple of weeks to a couple of months). However, delivery is not the same as release. Some projects may not see releasable functionality for a year or more. However, working software which can be demonstrated or used by the key stakeholders is the primary measure of progress. Iterative development is preferable primarily because it provides milestones that can't be concealed and an accurate measure of the progress can be ascertained (Agile Alliance, 2001).

Establishing Clear Goals & Objectives

Another key component in avoiding software project failure is to specify project goals and objectives that are precise, unambiguous, and

not overly ambitious in order to be able to derive complete, consistent, and realistic requirements on which the rest of the development will depend. This is often considered the most difficult aspect of software development (and one which Agile methodologies attempt to circumvent). Setting attainable and unambiguous goals is also a prerequisite to being able to determine when success has been attained and a project is complete.

Establishing clear goals and objectives up front also focuses the stakeholder groups on maintaining the same set of goals and objectives throughout the project (helping to eliminate another problem, "feature creep"). This does not mean to imply that changes cannot or do not happen (they will, and should be encouraged), but the goals and objectives of the project should remain constant. Failure to control for these changes is a significant reason for project failure and must be addressed in any robust software development methodology (something Agile methodologies achieve quite well, but traditional SDLC or "waterfall" methodologies are often criticized for handling poorly).

Requirements Analysis

> Nelson, Nelson & Ghodes (1998, p. 493) stated: "Structured methods can make a difference to the long term performance of software systems in many ways. Using structured methods can effect development and maintenance team efficiency and effectiveness. The overall quality and business value of the delivered system can be improved. User satisfaction with product attributes such as the format of information, the content of information, ease of use of the system, timeliness of information, and accuracy of information, as well as overall user satisfaction have also been shown to be impacted by structured methods."

Utilization of structured development methodologies can not only increase productivity of the developers but also their effectiveness. Utilization of structured methodologies can also reduce the impact of the differences in developers' abilities by formalizing the knowledge acquired by experienced developers (Yourdon, 1989; Nelson, Ghods, & Nelson, 1998).

Ewusi-Mensah (2003, p. 190) prescribes the following remedy to how to alleviate many of the failures which lead to project abandonment:

> "In general, superior designs are likely to lead to superior software if there are processes in place to guide the development. As always, comprehensive knowledge and understanding of the problem domain will be critical to a good design. Good design, though always crucial, will not automatically produce the quality software desired; managing the development process is every bit as important. Because software design is essentially a creative learning process, as the team becomes more familiar with the requirements, incremental steps can be taken to incorporate the various functionalities that need to be satisfied…"

In Agile methodologies, initial positive early results often mask the reality that the lack of an effective requirements analysis will make the complete system untenable. In one of the earliest examples of a significant software development project utilizing Agile methodologies, the Chrysler Comprehensive Compensation system, the first phase was by all accounts a complete success inviting the accolades of anyone who subsequently subscribed to the eXtreme Programming (XP) methodology. However, the lack of a significant requirements analysis for this complex system doomed it to eventual failure as it was never able to integrate completely with the rest of the system or scale to its target utilization. The system never saw its 2nd planned release even after 2 years of subsequent development (Garzaniti, Haungs, & Hendrickson, 1997; Wikipedia, 2006). As Boehm (2002, p. 67) states, "overfocus on early results in large systems can lead to major rework when the architecture doesn't scale up. In such cases, a good deal of planning will be necessary."

Empirical evidence shows that a comprehensive requirements analysis & design reduces overall implementation & maintenance costs. In their description of the "plan-driven" approaches to software development, Nerur and Balijepally (2007, p. 80) also state that "systematic problem decomposition is the key activity in analysis" which must be completed prior to synthesis of the knowledge in the system design. However, they also go on to state that simply decomposition of the existing processes is insufficient to successful

system analysis. Analysts must also seek to learn the process to fully understand it. The authors' own experiences show that comprehensive requirements analysis & design combined with component-based development can increase quality (which reduces maintenance costs), increases technology acceptance (by involving all of the key stakeholders in the requirements analysis), yet can still achieve incremental software delivery at "Internet Speed".

Stakeholder Commitment

A fundamental element in Agile methodologies as well as a key deficiency in many of the failed software development projects is stakeholder commitment and involvement in the software development process (particularly in the early phases of development). Ewusi-Mensah (2003, p. 48) states that "user commitment and involvement are critically important in helping to determine what the requirements of the system should be. End users' active involvement in the requirements phase is crucial for providing the software developers with the requisite information to enable the analysis and design to reflect fully the needs and circumstances of the user community." He goes on to state that particular attention needs to be paid to the "situations and circumstances" with which the stakeholders will use the software being developed.

Ewusi-Mensah (2003) goes on to define a process by which the development team communicates with the stakeholders which is similar to, but goes beyond, the framework suggested by Agile methodologies. In Ewusi-Mensah's model, which he refers to as the stakeholder-interaction model (SIM), a triangle of communications and interactions are established between the IS/technical staff, senior management, and the users group. This process seeks to maximize the desired features of the completed product within the constraints of the available resources and completion schedule.

The hard-line approach that proponents of current Agile methodologies take to relying on a small group of stakeholders (frequently a single point of contact) has the effect of placing blinders on the systems analysts and developers. It may simplify the process of eliciting requirements and validating progress on the implementation of the application, but at the expense of eliminating the input of other key stakeholders. Reliance on the single customer point of contact to

validate the correctness of the application within the rest of the organization is insufficient (Nerur, Mahapatra, & Mangalaraj, 2005). When pairing developers with individuals within the customer's organization, it often results in pluralistic decision-making environment which can lead to increased conflict, trust issues, and leaves other stakeholders in the organization at the mercy of the customer contact (Nerur, Mahapatra, & Mangalaraj, 2005).

Another issue with requiring such close contact with the customer during the implementation phase of the software development in the Agile methodologies is that it precludes the utilization of disparate teams which may have members located at various sites (Ambler, 2003) and would eliminate or hamper efforts to use outsourcing, virtual teams, etc.

Embrace Change

During the project lifetime, requirements change 25% or more (Boehm, 2002). Agile development is touted as being typically most beneficial when change occurs frequently whereas traditional methodologies are not responsive to change. Requirements are set at the beginning and are difficult to change and requested changes are typically not presented to the customer in usable form until after the project is completed.

Describing one of the guiding principles of Agile methodologies, the Agile Manifesto (Agile Alliance, 2001) states:

> "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. The growing unpredictability of the future is one of the most challenging aspects of the new economy. Turbulence—in both business and technology—causes change, which can be viewed either as a threat to be guarded against or as an opportunity to be embraced.
>
> Rather than resist change, the agile approach strives to accommodate it as easily and efficiently as possible, while maintaining an awareness of its consequences. Although most people agree that feedback is important, they often ignore the fact that the result of accepted feedback is change. Agile

methodologies harness this result, because their proponents understand that facilitating change is more effective than attempting to prevent it."

Use the Strengths of Every Developer

All of the Agile methodologies put a premium on having the absolute best personnel and work best with first-rate, versatile, disciplined developers and stakeholders who are highly skilled and highly motivated. Not only do you need developers with significant skills in a variety of areas (technical as well as interpersonal), but they also must possess exceptional discipline and motivation and be willing to work at a very high level with someone sitting beside them watching every move (Boehm, 2002). These characteristics are estimated to be present in only a small percentage of developers. As Boehm so eloquently puts it, over 49% of software developers are below average.

To be capable of being used within any organization, software development methodologies must be able to use developers, management, and users at all skill levels. Simply because a particular developer doesn't possess all of the requisite skills to be an effective Agile developer, doesn't mean that they can't contribute to the development process. Some of the best developers in the world have practically no interpersonal skills, but that doesn't make them ineffective if they can be allowed to work within their strengths.

Scale to Fit Any Project

Anything other than a trivial application (where triviality is determined based upon the complexity of the application, the number of users, as well as the strategic orientation of the application) requires rigorous software analysis and design. Because of this, utilization of current Agile methodologies doesn't scale well to large projects (Sliwa, 2002). When Constantine (2001) surveyed his colleagues who were proponents of "light" methods (which is how he describes Agile methodologies), they agreed that the Agile methodologies do not readily scale up beyond a certain point. Constantine stated that the general consensus was that 12 to 15 developers is the upper limit for most Agile projects. The primary reason for this problem is because of

the tightly coordinated teamwork which is needed for these methods to succeed.

Summary

MacCormack (2001) specifies the four factors which most contribute to the success of a software development project as:

- An early release of the evolving product design to the key stakeholders
- Iterative software development with frequent releases to the stakeholders and rapid feedback
- Teams comprised of people with broad-based experiences
- Major investment in the design of the product architecture

Both Agile and traditional SDLC-driven methods possess characteristics in which each clearly works better than the other for each of these factors and the other sources of failure demonstrated earlier. Hybrid approaches that combine both types of methodologies are feasible and necessary for just about every project type (Boehm, 2002). In the following section, we will propose a software development methodology which incorporates each of the facets of a methodology which were enumerated earlier to construct such a hybrid in order to increase the likelihood of the success of a software development implementation.

CHUNK

CHUNK is a software development methodology which incorporates many of the fundamental principles of the Agile methodologies within the traditional SDLC while removing many of the unnecessarily restrictive or superfluous components of the Agile methods (see FIGURE 2 below). This results in CHUNK combining a top-down and bottom-up approach to software development (where Agile methods typically use a bottom-up approach and the SDLC is predominately a top-down approach).

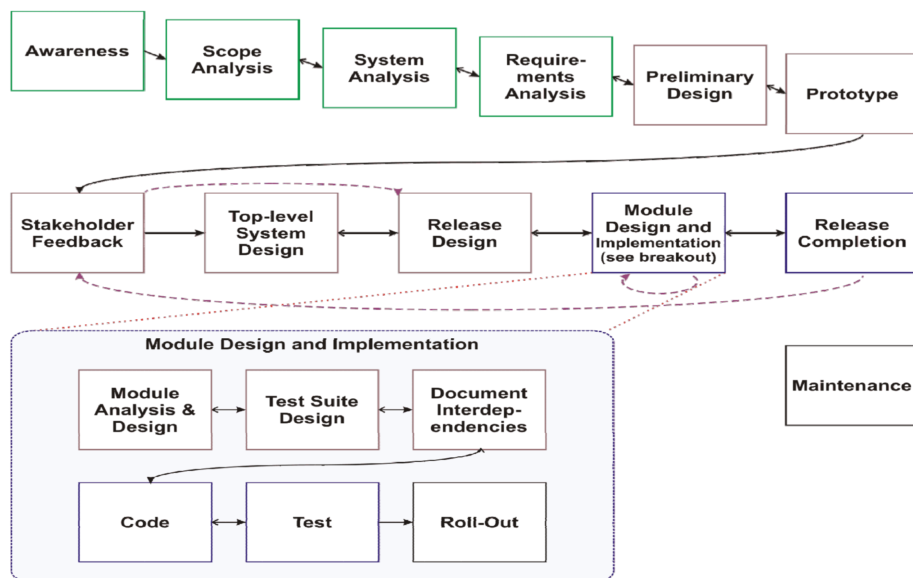
CHUNK can be more formally classified as a component-based or iterative software development methodology. Component-based software development has existed for decades, even prior to

methodological formalization. Although the methodology is language independent, a fundamental goal of its organization and design was to facilitate development and implementation through the utilization of object-oriented language constructs to develop the identified components.

Although the phases in this process are generally meant to be executed sequentially, or as otherwise indicated, unlike the traditional (although inaccurate) view of the "waterfall" methodology, it is expected that earlier phases can be revisited in the manners prescribed. Frequently, invocation of a subsequent step in the software development process results in new insights or requirements which must be incorporated into the project. The methodology, however, must be capable of handling these changes without significant disruption in the overall development process.

The subsequent sections describe in more detail each of the phases of the development process prescribed by CHUNK and illustrated in the framework in FIGURE 2.

FIGURE 2. CHUNK Development Framework



Awareness

Awareness of a problem to be solved can be arrived at through the initiative or innovation of an individual or group within an organization, a request for a unique solution from a customer, or the outgrowth of a strategic IS plan (Peffers, Gengler, & Tuunanen, 2003). Once an organization becomes aware of the need, an initial evaluation of the project needs to be made as to whether the organization will pursue the project. The project will then be setup in the Software Configuration Management system (SCM). Utilization of an SCM is necessary for maintaining documentation, tracking progress, and managing the code. All artifacts of the development process will be made accessible through the SCM.

Selection of the project team will also be accomplished at this point. In most situations, the ideal project team will be comprised of: a business analyst who should have an understanding of general business processes and how to integrate them with IT; a systems analyst who will have a general understanding of business processes, systems analysis, and development methodologies; a software engineer who will have a solid understanding of the software development process; and key stakeholders (e.g. product champions) who have an understanding of the domain of the system being developed and what the new system is to accomplish.

Scope Analysis

The scope analysis phase of the software development process is where the key stakeholders involved in the project are identified and the goals and objectives of the software project are established.

Stakeholder Analysis

Stakeholder analysis involves understanding who affects or is affected by the system under development (Robinson & Volkov, 1997). This will include not only the users of the system, but those individuals, groups, and organizations that are impacted by the system, those who are responsible for participating in the development of the system, senior management within the organization (Ewusi-Mensah, 2003), and other systems which must interact with the one under development.

In general, the users stakeholder group provides the most robust and ambitious requirements for the new system. Including the development group in the list of stakeholders helps to insure that the eventual requirements for the system are practical and can be accomplished with currently available technology and skills within the group. Including senior management in the list of stakeholders is necessary to insure that a practical limit is set to the scope of the project.

Goals and Objectives

Establishment of the goals and objectives of the project under development is necessary whether the project is being developed for use exclusively internally, for a client, or for commercial software. The goals and objectives of the software project will identify the major features or purposes of the software being developed, a recommended completion date, and an initial budget. This part of the scope analysis enables the project team to identify what the finished software project is intended to accomplish. It also establishes the objectives for determining when the project is completed.

System Analysis

The system analysis phase of the software development model focuses on the evaluation and analysis of existing processes, procedures, and systems. System analysis consists of collecting, organizing, and evaluating facts about the existing system and the environment in which it operates (including equipment, personnel, operating conditions, and the system's internal and external demands) (Couger, 1973). Common sources for this information can be found in the organization's existing documentation (e.g. procedure manuals), the current applications, corporate intranet, or from personnel.

This phase in the software development process is necessary in order to understand the processes and systems that are to be replaced so that the solicitation of requirements from the stakeholders can be done intelligently. Once a general understanding of how the process is currently implemented, determining what questions need to be asked of the stakeholders becomes much easier and more detailed information can be collected and potential gaps in what is required of the final system may be eliminated.

Requirements Analysis

Many methods have been used for arriving at the user requirements for the system under development. Unfortunately, in practice, one of the most frequently used is to take the initial project request with perhaps an individual's or small group's additional guidance as the complete requirements specification. This often results in a system which is less than adequate to meet the customer's needs, can't integrate with the related systems, can't scale to the necessary size, and ultimately fails or is infrequently used.

Requirements elicitation and analysis is generally considered one of the most difficult aspects of the traditional software development process (Browne & Rogich, 2001). Avoiding the comprehensive requirements analysis component of software development is one of the reasons Agile methodologies have achieved their allure, but which ultimately reduces the likelihood of success for the project. If properly executed, requirements analysis can not only result in developing systems that the users are more satisfied with, but can also ease the development effort.

An additional benefit to a comprehensive requirements elicitation and analysis is that by involving the stakeholders in the development process, the future buy-in into the completed system becomes much more likely. One of the ways that projects have been termed as failures is if the software is unused after completion. This possibility can be significantly reduced by getting the stakeholders to participate in its development and thus increase the buy-in to the final product.

Requirements analysis also needs to identify how the different stakeholders interact. Interactions occur whenever 2 or more requirements are dependent upon each other. Negative interactions (conflicts) occur when one requirement interferes with one of the other requirements, often from another stakeholder (Robinson & Volkov, 1997). A process for dealing with these conflicts is also necessary to enable successful project completion.

Three methods for eliciting and analyzing requirements specifications from the stakeholders are interviews & surveys, use case analysis, and task shadows. These are only 3 of the possible methods that can be used within Chunk, but together they allow for the elicitation of the most robust and complete requirements of the system under development.

Interviews and Surveys

One of the most misunderstood, misapplied, and mistake-prone methods of user requirements elicitation is the use of the interview. Too often, questions in the interview process do not elicit a complete view of the system to be developed. A comprehensive interview script needs to include questions to elicit responses to the who, what, when, where, how, and why of the system being developed. Browne and Rogich (2001) detail a process which will allow the analysts to educe customer requirements in a generalized format which can be easily integrated into the CHUNK process.

Use Cases

Use Cases describe the system's behavior under various conditions as it responds to a request from one of the stakeholders. In the development of a use case, the stakeholder describes an interaction with the system to accomplish some goal. The system then delivers an appropriate response based upon the input of the stakeholder (Cockburn, 2000). Use cases are helpful not only as a method of eliciting stakeholder requirements, but provide important input to the developers who can use the analysis directly in the implementation of the system, the technical writers who must describe how to use the system, and the testers who can utilize the Use cases in the development of their test suites (Bittner, 2002).

Task Shadowing

Another concept which can be used as a method for eliciting user requirements is the use of Task Shadows. With this technique, users of the existing system for performing the tasks to be implemented in the new system (whether it is an existing IS or manual process) will be shadowed by someone who will keep track of each of the steps the user goes through while performing the task. One of the problems typically associated with interviews and Use Cases (as well as other techniques for eliciting user requirements) is that much of the knowledge about how a task is performed is tacit (e.g., the user isn't even aware of what they go through to perform a given task). In many situations, users have had to develop workarounds for deficiencies in the current processes that they may not even be aware of. By shadowing users as they are performing the tasks, seemingly insignificant details about how the task is accomplished can be

documented and implemented in the new system. If personnel cannot be allocated to each group of users, having the users keep a log of each step they go through while performing the tasks can elicit some of the details they may not have otherwise considered important.

Preliminary Design

Once the stakeholder requirements have been collected, analyzed and input into the SCM, development of an initial design of the system can begin. This is a preliminary conceptual design of the system which will describe the inputs and outputs of the system and the procedures used in processing those inputs to produce the required outputs.

This is the first phase of the development process which begins to look at the underlying architecture of the system under development. Major components, processes, or modules of the system will be identified in this phase which will be further refined once initial stakeholder feedback is received. To better enable a consistent description of the system for subsequent phases, design refinements, input into a modeling application, and future reuse, an Architecture Description Language (ADL) such as the type proposed by Dashofy, Hoek & Taylor (2005) should be used to define the relationships between the modules.

In addition to the initial component-based architecture which is defined in this phase, the Human Computer Interaction (HCI) components of the application will also be developed. HCI includes all aspects of the system that impact its users or other stakeholders. Although the user interface is one of the most noticeable components of the HCI and is key to the systems eventual acceptance, it is not the only part of the HCI development (Zhang, Carey, Te'eni, & Tremaine, 2005).

During the preliminary design phase of the software development process, issues relating to conflicts in requirements analysis will frequently be identified. As much as possible, analysts and developers should document reasons for deviations from, or omissions of requests for, features from the requirements analysis to ensure that subsequent concerns about these alterations can be properly justified.

Prototype

Prototyping of an application can take several forms: illustrative (screen shots), simulated (data flow), functional (limited subset of function), and evolutionary (Connors, 1992). For CHUNK, this phase of the process will involve the first 2 forms of prototyping where the initial User Interface (UI) design and process flow diagrams will be developed. Functional and evolutionary forms of the prototyping method should be avoided as developers often will exhibit resistance to discarding the work that went into the development of these functional prototypes (Ewusi-Mensah, 2003).

Several benefits to developing prototypes in this way have been identified. Stakeholders in the system may not know exactly what they want out of the system even after the requirements analysis and the prototype provides the user with additional information with which they can make a more informed decision as to whether or not the application under development will meet their needs. Additionally, the prototyping phase can more accurately define the scope of the ensuing system. Changes made at this phase can be 6-10 times less expensive than once the system has entered the implementation phase (Connors, 1992).

Stakeholder Feedback

Although nothing in this software development process precludes the design team from reiterating or revising any of the previous phases (e.g. interviews and use case analysis in the requirements analysis phase may identify new stakeholders which may require revisiting the Scope Analysis and subsequent phases), all of the previous phases are designed to be completed in sequence. The Stakeholder Feedback phase of the methodology, however, is intended to be executed multiple times throughout the system development lifecycle.

The initial pass through the stakeholder feedback phase is designed to elicit stakeholder feedback to the preliminary design and prototypes that have been developed. This process is intended to raise any additional issues or concerns with the initial design and scope of the application. Discussion will also revolve around the conflicts identified in the requirements analysis and preliminary design phases to address the decisions made by the design team regarding which

elements were included and why some elements may have had to be excluded from the design. Further refinement of the prototypes will also be focused on in the initial pass through the stakeholder feedback phase. Prioritization of the components (functions and tasks to be implemented in a release) to be completed from the stakeholder's perspective will also be necessary.

Subsequent iterations of the stakeholder feedback phase will address the validation of the completed releases of the software project as well as requests for modifications to be made to the requirements, the completed releases, and component prioritization. It is imperative to the success of a project that as many stakeholder groups as possible be represented in the stakeholder feedback sessions. The frequency of these sessions relates to the size and scope of each of the release design and is also positively related to the project success (e.g. the smaller the release, the more frequent the stakeholder feedback iterations, and consequently the greater chance of success for the project).

Because of the intensity of communications among stakeholders in this phase of the development process, methods for coordinating these activities need to be used. One of these methods could include the use of Rapid Application Development (RAD) sessions. RAD techniques seek to bring order to the relative chaos of the system design at this phase in the development process. Its primary purpose is to take the initial system concept and turn it into a working system design which adds value to the business operation in a relatively short period of time (Howard, 2002) while having the input of the primary stakeholder groups.

If this is not the initial iteration of the stakeholder feedback phase and no modifications to the plan have been identified, the development process can proceed to the release design phase.

Top-level System Design

Upon completion of the initial iteration of the stakeholder feedback phase of the development lifecycle, the top-level system design can proceed. At this point, the initial requirements of the system from the stakeholder's perspective have been elicited and input into the SCM, the requirements have been coalesced into a validated requirements specification, and a prototype of the user interface and data flow have been completed and also input into the SCM. With this comprehensive

body of information which has been reviewed and verified by the stakeholders, the system design can proceed with relative speed. Possessing a comprehensive understanding of the needs of the stakeholders and prototypes of the system to be developed eliminates much of the uncertainty which often accompanies a system design.

This initial iteration of the top-level system design involves developing the module definitions, interfaces and communications between modules, as well as hardware and software requirements and the environment in which the application is to be run. Subsequent iterations of this phase will revise the top-level design if necessary to accommodate requests for changes to the system that were expressed during the stakeholder feedback phases.

The modules are those segments of the application which can be developed independently and which are designed to perform a specific task. In object-oriented systems, a module would be a class or a tightly defined set of classes in a particular hierarchy. In procedurally developed systems, a module may be implemented as a code library to implement a discrete task or function.

In order to improve the reliability and ease of development, modules should be designed to minimize tight coupling and interdependencies. The more independently modules can be designed, the easier the tasks of implementation, testing, and maintenance become.

Module Prioritization

Outside of the component prioritization in the stakeholders feedback phase, module prioritization must also be specified as certain modules may be used across multiple components or within multiple releases. As a general guideline for determining the priority of modules implemented early in the software development, the following criteria should be considered:

Modules with larger number of interdependencies should be implemented earliest in the software development (they require the greatest amount of specification, documentation, and testing)

- Modules which provide UI capabilities
- Initial product functionality modules
- Remaining modules required for the current release

Release Design

The release design phase is primarily focused on defining the components and modules to be implemented in the release and defining the criteria used for testing its implementation. The test suites developed for the release will need to be added to the SCM and should be based upon the use case analysis and prototypes developed during the analysis and design of the software application. This phase will also determine the personnel needed to perform the tasks necessary for the implementation and testing of the release. Modules to be implemented into the release will be added to the SCM at this point as well.

Module Design and Implementation

Because the implementation of the project is isolated from the processes involved in the analysis, design, and stakeholder feedback, the Module Design and Implementation phase can be implemented at any location which has accessibility to the SCM. This enables the utilization of outsourcing of the programming of the application and the utilization of programmers who may be very skilled in coding, but who may lack skills necessary for other development methodologies. Since the programming is typically a significant cost associated with application development, this can have the effect of significantly lowering the overall costs of the project.

Additionally, since the definitions and interfaces of the modules incorporated into a release and all of the analysis and design documentation is available through the SCM, work on the modules can generally be handled concurrently. The only exception may be those modules which have been identified as having a high degree of interdependency which may have to be completed before work on other modules can begin.

Common to each of the steps involved with module design and implementation is extensive documentation within the code and the SCM. Fundamental components of the documentation include (but are not limited to):

Change management (documenting all changes to the software, who made the changes, what the original code included, purpose of the change)

Interdependencies (all modules that use the module under development MUST be documented in the module heading. This provides a test suite to ensure that changes to the module can be tested properly and interface changes can be readily adapted to the rest of the system)

Many of these documentation requirements can be automated through the use of the SCM

Additionally, proper documentation, module-level analysis and design, and comprehensive testing of individual components will allow developers to maximize the reusability of their code developed within this framework. Warehouses of the developed modules with their corresponding documentation will make the modules available to other developers and for other projects maximizing their reusability.

For each module to be developed for the current release of the project, the following processes will need to be completed:

Module Analysis and Design (MAD)

In this process, a bottom-up approach to the system development process will be used. A more comprehensive design of the activities and functions involved in the module will be completed (the Top-Level System Design only defines the principle functions of the modules and how they will communicate with other modules).

Determine test suite for module

Before the coding of the module is begun, generating a test suite for the module based upon the module description and the use case analysis will need to be defined. Careful attention needs to be paid especially to the inputs and outputs through the defined interfaces of the module as well as bounds checking on all constructs. The test suite must also be added to the SCM to ensure that when this module needs to be re-tested, the complete test suite is readily available.

Document module interdependencies

Although primary interdependencies have already been identified, there may be additional module interdependencies discovered during the implementation of the modules. Proper documentation of interdependencies is necessary to ensure that when changes to a module occur, all other modules which depend on the changed module

can be re-tested. The SCM should be able to automatically determine and add the dependency hierarchy to the module definition.

Code module

Many methods have been proposed to increase the productivity or correctness of the actual code used to implement a software system. Pair programming is one of these methods which has shown some promising results and can be used, but it is not mandated.

Test module

Once the initial module programming is completed, the defined tests for the module will need to be executed. Any additional tests identified during the coding or testing of the module will also need to be added to the SCM and executed.

Roll out module

Once the testing is complete successfully and all appropriate documentation and test suites have been added to the software configuration management system, the module can be marked as complete and stored in the SCM.

Release Completion

Once all of the modules that have been identified as a part of the release have been completed, the modules will be combined into a single release and the required component and functional testing of the release can begin. Any errors or deficiencies can be identified and communicated to the developers of the module and the release coordinator. Once an acceptable level of success has been achieved, the release can be marked as completed.

Once a release is complete, it will be rolled out for utilization and work can begin on the next release of the project. At this point, the process will return to the phase to elicit stakeholder feedback (if applicable) for an acceptance test and requests for modifications to the current release or future releases. If change requests occur, they will be rolled into the project through the re-iteration of the Top-level System Design.

Maintenance

Once the final release has been completed and final acceptance tests have been performed by the stakeholders, the project is complete and will move into the Maintenance phase. In addition to performing bug fixes and adding functionality to the system, once the project moves into the Maintenance phase, a post-mortem of the development process will be completed. The post-mortem requires the artifacts of the development process (which are non-existent in the Agile methodologies) to learn from and improve upon future development projects.

CONCLUSION

It has been established that there is a need for a proper framework for software development and that this framework should include:

- support for an iterative development process,
- clear goals and objectives which need to be expressly defined,
- a comprehensive requirements analysis,
- stakeholders need to be identified and committed to the process,
- a formal change process needs to be used,
- the skills of everyone involved in the implementation need to be employed,
- the process should be scalable.

CHUNK is a methodology which has been proposed to address each of these issues in order to achieve a greater probability of the successful completion of the application development process. This methodology incorporates many of the benefits of Agile development techniques into the SDLC with much more rigorous Analysis and Design and revolves around component-based incremental releases.

No development methodology can guarantee success of a software project implementation. However, lack of an appropriate development methodology may lead to the failure of the project before it even begins. Project failure in itself would not result in a disaster for the organization so long as the failure occurs before significant resources have been expended towards the project, however. Consequently, if a project is to fail, identifying the failure early on in the development

process prior to implementation would be the ideal situation. With the comprehensive analysis and design and stakeholder feedback which are primary components of CHUNK, it is more likely that a project may be identified as untenable before the expenditure of the much larger number of resources for its implementation.

Another primary feature in the design of Chunk is that the implementation of the project is isolated from the processes involved in the analysis, design, and stakeholder feedback (unlike other Agile methodologies). Consequently, distribution of the majority of the work in the implementation of the software project is fairly uncomplicated. This allows the utilization of programmers which do not have the significant interpersonal skills required by Agile methodologies and which may be located at diverse locations.

Consequently, utilization of CHUNK can contribute to the rapid delivery of software rolled out in phases, lower the costs associated with the implementation of the software, and provide the greatest likelihood of project success.

REFERENCES

Agile Alliance. (2001). *Manifesto for Agile software development.* Retrieved May 19, 2008, from The Agile Manifesto: http://www.agilemanifesto.org

Ambler, S. W. (2003, October 1). *Chicken Little was right*. Retrieved May 19, 2008, from Dr. Dobb's Journal: http://www.ddj.com/architect/184415045

Baskerville, R., & Pries-Heje, J. (2004). Short cycle-time systems development. *Information Systems Journal , 14* (3), 237-164.

Bennett, K. (1991). Automated support of software maintenance. *Information & Software Technology , 33* (1), 74-85.

Bittner, K. (2002). *Use Case Modeling.* Boston: Addison-Wesley Longman Publishing Co., Inc.

Boehm, B. (2002). Get ready for Agile methods, with care. *Computer , 35* (1), 64-69.

Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. *Computer , 34* (1), 135-137.

Browne, G. J., & Rogich, M. B. (2001). An empirical investigation of user requirements elicitation: Comparing the effectiveness of

prompting techniques. *Journal of Management Information Systems , 17* (4), 223-249.

Cockburn, A. (2000). *Writing Effective Use Cases* (1st Edition ed.). Boston: Addison-Wesley Longman Publishing Co., Inc.

Connors, D. T. (1992). Software development methodologies and traditional and modern information systems. *ACM SIGSOFT Software Engineering Notes , 17* (2), 43-49.

Constantine, L. (2001, June). Methodological Agility. *Software Development* , pp. 67-69.

Couger, J. D. (1973). Evolution of business system analysis techniques. *ACM Computing Surveys , 5* (3), 167-198.

Dashofy, E. M., Hoek, A. v., & Taylor, R. N. (2005). A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology , 14* (2), 199-245.

Ewusi-Mensah, K. (2003). *Software Development Failures.* Cambridge: MIT Press.

Garzaniti, R., Haungs, J., & Hendrickson, C. (1997). Everything I need to know I learned from the Chrysler payroll project. *Conference on Object Oriented Programming Systems Languages and Applications* (pp. 33-38). Atlanta: ACM.

Glass, R. L. (2003). A mugwump's-eye view of Web work. *Communications of the ACM , 46* (8), 21-23.

Glass, R. L. (2006). The Standish report: does it really describe a software crisis? *Communications of the ACM , 49* (8), 15-16.

Hartman, D. (2006). *Interview: Jim Johnson of the Standish Group*. Retrieved October 7, 2006, from http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS

Howard, A. (2002). Rapid Application Development: rough and dirty or value-for-money engineering? *Communications of the ACM , 45* (10), 27-29.

MacCormack, A. (2001). Product-development practices that work: How Internet companies build software. *MIT Sloan Management Review , 42* (2), 75-84.

Nelson, K. M., Ghods, M., & Nelson, H. J. (1998). Measuring the effectiveness of a structured methodology: A comparative analysis. *Proceedings of the Thirty-First Annual Hawaii*

International Conference on System Sciences. 6, pp. 492-499. Hilo, HI: IEEE Computer Society.

Nerur, S., & Balijepally, V. (2007). Theoretical reflections on Agile development methodologies. *Communications of the ACM , 50* (3), 79-83.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to Agile methodologies. *Communications of the ACM , 48* (5), 72-78.

Peffers, K., Gengler, C. E., & Tuunanen, T. (2003). Extending critical success factors methodology to facilitate broadly participative information systems planning. *Journal of Management Information Systems , 20* (1), 51-85.

Poole, D. (2006). Breaking the major release habit. *Queue , 4* (8), 46-51.

Robinson, W. N., & Volkov, S. (1997). A meta-model for restructuring stakeholder requirements. *Proceedings of the 19th international conference on Software engineering* (pp. 140-149). Boston: ACM.

Sliwa, C. (2002, March 14). *Agile programming techniques spark interest*. Retrieved March 12, 2007, from ComputerWorld: http://www.computerworld.com/action/article.do?command=printArticleBasic&articleId=69079

The Hackett Group. (2003). *Profile of World-Class IT.* London: The Hackett Group.

Wikipedia. (2006). *Chrysler Comprehensive Compensation System*. Retrieved October 23, 2006, from http://en.wikipedia.org/wiki/Chrysler_Comprehensive_Compensation_System

Yourdon, E. (1989). *Modern Structured Analysis.* Upper Saddle River, NJ: Yourdon Press.

Zhang, P., Carey, J., Te'eni, D., & Tremaine, M. (2005). Integrating human-computer interaction development into the systems development life cycle: a methodology. *Communications of the AIS , 15*, 512-543.

RECEIVED: February 10, 2008
REVISED: March 23, 2008
ACCEPTED: April 15, 2008