SOFTWARE ENGINEERING METHODOLOGY FOR DEVELOPING SYSTEM SOFTWARE

Ezekwe Chinwe Genevra[1], Ugah John Otozi[2], Okoh Felix Jigiedous[3]
Edeh Eucharia Ujunwa[4] ,Okoye Joy Anulika[5]

[1] Electronics Development Institute, Awka National agency for Science and Engineering Infrastructure (NASENI)Federal ministry of Science and Technology, [2]Department of Computer Science, Ebonyi State University, Abakiliki, [3] Federal College of Education, Asaba, [4] Project Development Institute, Enugu, [5] Department of Electronics and Electrical Engineering, Chukwuemeka Odumegwu Ojukwu University, Uli, NIGERIA.

[1]norakingchi@yahoo.com,[2]ugah@gmail.com [3] jigiefelix@yahoo.com
[4] ujuedeh@yahoo.com, joynuli.ok98@gmail.com

ABSTRACT

A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of designing software. Others apply project management techniques to designing software. Without effective project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, it is effective project management that appears to be lacking. The Microsoft solution framework came to the rescue of these long lasting problems using its best component called MSF for Capability Maturity Model Integration Process Improvement methodology (MSF4CMMI) which is a process improvement approach that provides organizations with the essential elements of continuous process improvement resulting in a reduced Software Development Life Cycle, improved ability to meet the cost and schedule targets, building products of high quality. The MSF4CMMI has extended the MSF4ASD guidance with additional formality, reviews, verification and audit. This results in a Software Engineering Products that relied on process and conformance to process rather than relying purely on trust and the ability of the individual team members.

Key words: system software, software methodology, software engineering and Microsoft solution framework

INTRODUCTION

In software engineering, a software development methodology (also known as a system development methodology, software development life cycle, software development process, software process) is a division of the software development work into distinct phases or activities, with the intent of better planning and management. A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system by Alan (2004). A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. It is often considered as a subset of the systems development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application. Common methodologies include: a) waterfall, b) prototyping, c) iterative and incremental development, d) spiral development, e) rapid application development, and f) Extreme programming.

Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a

ISSN: 2223-9944, e ISSN: 2223-9553

specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model. One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.

HISTORICAL BACKGROUND

The software development methodology (also known as SDM) framework did not emerge until the 1960s. According to Elliott (2004) the systems development life cycle (SDLC) can be considered to be the oldest formalized methodology framework for building information systems. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle from inception of the idea to delivery of the final system, to be carried out rigidly and sequentially"(Kuhn, 1989) within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".(Kuhn, 1989) Methodologies, processes, and frameworks range from specific proscriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process.

RELATED LITERATURES

Alan (2004) defines Systems software as the computer software designed to operate and control the computer hardware and to provide a platform for running application software. In some publications, the term system software also includes software development tools (like compiler, linker or debugger). In contrast to system software, software that allows users to do things like create text documents, play games, listen to music, or web browsers to surf the web are called application software. Exceptions could be e.g. web browsers such as Internet Explorer where Microsoft argued in court that it was system software that could not be uninstalled. Later examples are Chrome OS and Firefox OS where the browser functions as the only user interface *and* the only way to run programs (and other web browser cannot be installed in their place), then they can well be argued to *be* (part of) the operating system and then system software. System software can be separated into four different categories, operating systems, utility software, library program and translator software (Edward,2003).

Operating System

An operating system is a set of programs that manage computer hardware resources and provide common services for application software as defined by Edward, (2003). The operating system, like z/OS, Microsoft Windows, Mac OS X and Linux, allows the parts of a computer to work together by performing tasks like transferring data between memory and disks or rendering output onto a display device. It also provides a platform to run high-level system software and application software. The operating system is the most important type of system software in a computer system. Without an operating system, a user cannot run an application program on their computer (unless the application program is self booting).

ISSN: 2223-9944, e ISSN: 2223-9553

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input/output and main memory management, the operating system acts as a middleman between application programs and the computer hardware, although the application code is usually executed directly by the hardware it will frequently call by the OS or be interrupted by it. Operating systems can be found on almost any device that contains a computer, from mobile phones and video game consoles to supercomputers and web servers. The operating system is made up of these main parts by Edward (2003);

i. **A *kernel*** is the core part of the operating system that defines an Application program Interface (API) for applications programs (including some system software) and an interface to device drivers.

ii. ***Device drivers*** such as computer BIOS and device firmware provide basic functionality to operate and control the hardware connected to or built into the computer.

iii. **A *user interface*** "allows users to interact with a computer. Since the 1980s the graphical user interface (GUI) has been perhaps the most common user interface technology. The command-line interface is still a commonly used alternative.

Computer-aided software engineering

Computer-aided software engineering (CASE), in the field software engineering is the scientific application of a set of tools and methods to software which results in high-quality, defect-free, and maintainable software products. It also refers to methods for the development of information systems together with automated tools that can be used in the software development process as defined by Royce, (2012). The term "computer-aided software engineering" (CASE) can refer to the software used for the automated development of systems software, i.e., computer code. The CASE functions include analysis, design, and programming.

CASE tools automate methods for designing, documenting, and producing structured computer code in the desired programming language (Royce, 2012).

Two key ideas of Computer-aided Software System Engineering (CASE) are:

i. Foster computer assistance in software development and software maintenance processes and

ii. An engineering approach to software development and or maintenance.

Typical CASE tools exist for configuration, management, data modeling, model transformation, refactoring, source code generation, and Unified Modeling Language

System Software Development

System Software development is the computer programming, documentation, testing, and bug fixing involved in creating and maintaining applications and frameworks involved in a software release life cycle and resulting in a software product as defined by Whitten, et al (2003). The term also refers to a process of writing and maintaining the source code, but in a broader sense of the term it includes all that is involved between the conception of the desired software through to the final manifestation of the system software, ideally in a planned and structured process. Therefore, system software development may include research, new

ISSN: 2223-9944, e ISSN: 2223-9553

development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in system software products.

Software can be developed for a variety of purposes, the three most common being;

i. to meet specific needs of a specific client/business (the case with custom system software),

ii. to meet a perceived need of some set of potential users (the case with commercial and open source system software), or

iii. for personal use in the case of embedded system software (e.g. a scientist may write software to automate a mundane task).

Embedded software development

Embedded software development, that is, the development of embedded software that is codes (in C or Assembly language) that are build and burn using a universal programmer into the microcontrollers, PICs(programmable integrated circuits), PLDs (programmable logic devices) such as for use in controlling consumer products, requires the development process to be integrated with the development of the controlled physical product. System software underlies applications and the programming process itself is often developed separately before coupling with other components.

The need for better quality control of the software development process has given rise to the discipline of software engineering, which aims to apply the systematic approach exemplified in the engineering paradigm to the process of system software development.

System Software Development Activities

In system software development, there are some activities the developers must undergo depending on the software development approach and the development life cycle, we will highlight on five activities in this section.

Identification of need (Problem Identification)

The sources of ideas for system software products are legion. These ideas can come from market research including the demographics of potential new customers, existing customers, sales prospects who rejected the product, other internal system software development staff, or a creative third party. Ideas for system software products are usually first evaluated by marketing personnel for economic feasibility, for fit with existing channels distribution, for possible effects on existing product lines, required features, and for fit with the company's marketing objectives. In a marketing evaluation phase, the cost and time assumptions become evaluated. A decision is reached early in the first phase as to whether, based on the more detailed information generated by the marketing and development staff, the project should be pursued further. (Whitten, et al 2003)

Planning

Planning is an objective of each and every activity, where we want to discover things that belong to the project. An important task in creating a system software program is extracting the requirements or requirements analysis.(Whitten, et al 2003) Customers typically have an abstract idea of what they want as an end result, but do not know how to create system *software* that will execute their ideas. Skilled and experienced software engineers recognize incomplete, ambiguous, or even contradictory requirements at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

ISSN: 2223-9944, e ISSN: 2223-9553

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

Designing

Once the requirements are established, the design of the software can be established in a software design document. This involves a preliminary, or high-level design of the main modules with an overall picture (such as a block diagram) of how the parts fit together. The language and hardware components should all be known at this time. Then a detailed or low-level design is created, perhaps with prototyping as proof-of-concept or to firm up requirements.

Implementation, testing and documentation

Implementation is the part of the process where software engineers actually program the code for the project.

Software testing is an integral and important phase of the software development process. This part of the process ensures that defects are recognized as soon as possible.

Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an API, be it external or internal. The software engineering process chosen by the developing team will determine how much internal documentation (if any) is necessary. Plan-driven models (like Waterfall model) generally produce more documentation than Agile models.

Deployment and maintenance

Deployment starts directly after the code is appropriately tested, approved for release, and sold or otherwise distributed into a production environment. This may involve installation, customization (such as by setting parameters to the customer's values), testing, and possibly an extended period of evaluation(Whitten, et al 2003).

Software training and support is important, as software is only effective if it is used correctly.

Maintaining and enhancing software to cope with newly discovered faults or requirements can take substantial time and effort, as missed requirements may force redesign of the software.

SYSTEM SOFTWARE DEVELOPMENT METHODOLOGY

A software development methodology (also known as a software development process, model, or life cycle) is a framework that is used to structure, plan, and control the process of developing information systems as defined by Arcisphere (2012). A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. There are several different approaches to software development: some take a more structured, engineering-based approach to developing business solutions, whereas others may take a more incremental approach, where software evolves as it is developed piece-by-piece. One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.

ISSN: 2223-9944, e ISSN: 2223-9553

Most methodologies share some combination of the following stages of software development (Arcisphere, 2012):

i. Analyzing the problem
ii. Market research
iii. Gathering requirements for the proposed business solution
iv. Devising a plan or design for the software-based solution
v. Implementation (coding) of the software
vi. Testing the software
vii. Deployment
viii. Maintenance and bug fixing

These stages are often referred to collectively as the software development lifecycle (SDLC).

System software Development Methodology Approaches

Several software development approaches have been used since the origin of information technology, in two main categories. Typically **an approach** or a **combination of approaches** as in fig. 1 is chosen by management or a development team. Different approaches to software development may carry out these stages in different orders, or devote more or less time to different stages. The level of detail of the documentation produced at each stage of software development may also vary. These stages may also be carried out in turn (a "**waterfall" based approach**), or they may be repeated over various cycles or iterations (a more **"extreme" approach**). The more extreme approach usually involves less time spent on planning and documentation, and more time spent on coding and development of automated tests. More "extreme" approaches also promote continuous testing throughout the development lifecycle, as well as having a working (or bug-free) product at all times as stated by Arcisphere (2012).

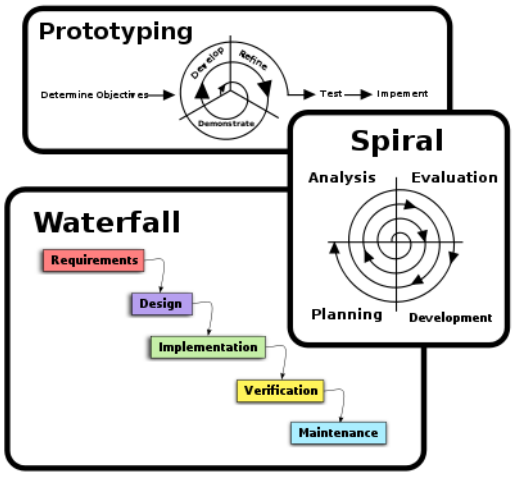

Fig. 1. Combined system software development methodology (Arcisphere (2012).

More structured or "waterfall" based approaches attempt to assess the majority of risks and develop a detailed plan for the system software before implementation begins, and avoid significant design changes and re-coding in life cycle planning.

ISSN: 2223-9944, e ISSN: 2223-9553
www.journals.savap.org.pk

Software Engineering Methodology for Developing System Software

In the development of system software, Microsoft Inc is the most popular developer. In this sector we will lay emphasis on the methodology they are presently using for system software development which is called Microsoft Solutions Framework (MSF). It is a set of principles, models, disciplines, concepts, and guidelines for delivering information technology solutions from Microsoft by McConnell, (2004). MSF is not limited to developing system software only; it is also applicable to other IT projects like deployment, networking or infrastructure projects. MSF does not force the developer to use a specific methodology (Waterfall, Agile, etc) but lets them decide what methodology to use.

Goals of Microsoft solution Framework

i. Microsoft Solutions Framework (MSF) is a set of software engineering processes, principles, and proven practices intended to enable developers to achieve success in the software development life cycle (SDLC) (Whitten, 2003).

ii. MSF provides an adaptable guidance, based upon experiences and best practices from inside and outside of Microsoft, to increase the chance of successful delivery of an information technology solution to the customer by working fast, decreasing the number of people on the project team, averting risk, while enabling high quality results.

Components of Microsoft Solution Framework

The Microsoft Solution Framework version 4.0 is a combination of a metamodel which can be used as a base for prescriptive software engineering processes, and two customizable and scalable software engineering processes (McConnell, 2004). The MSF meta model consists of

i. Foundational principles,

ii. a team model and cycles and

iii. Iterations.

MSF 4.0 provides a higher-level framework of guidance and principles which can be mapped to a variety of prescriptive process templates. It is structured in both descriptive and prescriptive methodologies. The descriptive component is called the **MSF 4.0 metamodel,** which is a theoretical description of the SDLC best practices for creating SDLC methodologies. Microsoft is of the opinion that organizations have diverging dynamics and contrary priorities during their software development; some organizations need a responsive and adaptable software development environment, while others need a standardized, repeatable and more controlled environment. To fulfill these needs, Microsoft represents the metamodel of MSF 4.0 in two prescriptive methodology templates that provide specific process guidance, named

i. Microsoft Solutions Framework for Agile Software Development (MSF4ASD) and

ii. Microsoft Solutions Framework for Capability Maturity Model Integration Process Improvement (MSF4CMMI).

Note that these software engineering processes can be modified and customize to the preferences of organization, customer and project team.

Foundational Principles

The following are the eight foundational principles, which form the backbone for the other models and disciplines of MSF:

ISSN: 2223-9944, e ISSN: 2223-9553

i. Foster open communication
ii. Work towards a shared vision
iii. Empower team members
iv. Establish clear accountability and shared responsibility
v. Focus on delivering business value
vi. Stay agile, expect change
vii. Invest in quality
viii. Learn from all experiences

MSF Models

MSF consists of two models(McConnel, 2004);

- the MSF team model and
- the MSF Governance Model
- ***MSF Team Model***

This describes the role of various team members in a software development project.

The members of this team would be:

- **Product Management**: Mainly deals with customers and define project requirements, also ensures customer expectations are met.
- **Program Management**: Maintains project development and delivery to the customer
- **Architecture**: Responsible for solution design, making sure the solution design optimally satisfies all needs and expectations
- **Development**: Develops according to the specifications.
- **Test**: Tests and assures product quality
- **Release/Operations**: Ensures smooth deployment and operations of the software
- **User Experience**: Supports issues of the users.

One person may be assigned to perform multiple roles. MSF also has suggestion on how to combine responsibilities such as the developer should not be assigned to any other role.

MSF Governance Model

This describes the different stages in processing for a project. The MSF Governance Model has five overlapping tracks of activity, each with a defined quality goal. These tracks of activity define what needs to be accomplished and leave how they are accomplished to the team selected methodology. For instance, these tracks can be small in scope and performed quickly to be consistent with an Agile methodology, or can be serialized and elongated to be consistent with a Waterfall methodology.

Tracks of activity:

- **Envision** - think about what needs to be accomplished and identify constraints
- **Plan** - plan and design a solution to meet the needs and expectations within those constraints

ISSN: 2223-9944, e ISSN: 2223-9553

- **Build** - build the solution
- **Stabilize** - validate that the solution meets the needs and expectations... "synch and stabilize"
- **Deploy** - deploy the solution

MSF Project Management Process

- Integrate planning and conduct change control,
- Define and manage the scope of the project,
- Prepare a budget and manage costs,
- Prepare and track schedules,
- Ensure that right resources are allocated to the project,
- Manage contracts and vendors and procure project resources,
- Facilitate team and external communications,
- Facilitate the risk management process,
- Document and monitor the team's quality management process(Whitten, 2003).

MSF for Agile Software Development methodology

The MSF for Agile Software Development (MSF4ASD) is intended to be a light weight, iterative and adaptable process. The MSF4ASD uses the principles of the agile development approach formulated by the Agile Alliance(McConnell, 2004). The MSF4ASD provides a process guidance which focuses on the people and changes. It includes learning opportunities by using iterations and evaluations in each iteration.

MSF for Capability Maturity Model Integration Process Improvement methodology

The MSF for Capability Maturity Model Integration Process Improvement (MSF4CMMI) has more artifacts, more processes, more signoffs, more planning and is intended for projects that require a higher degree of formality and ceremony (McConnell, 2004).

The MSF4CMMI is a formal methodology for software engineering. Capability Maturity Model was created at the Software Engineering Institute of Carnegie Mellon University, and is a process improvement approach that provides organizations with the essential elements of continuous process improvement resulting in a reduced SDLC, improved ability to meet the cost and schedule targets, building products of high quality. The MSF4CMMI has extended the MSF4ASD guidance with additional formality, reviews, verification and audit. This results in a Software Engineering Products that relies on process and conformance to process rather than relying purely on trust and the ability of the individual team members. The MSF4CMMI has more mandatory documents and reports than the agile version, and this formal development process reduces risk on large software projects and provides a measurable status.

DISCUSSION

There are significant advantages and disadvantages to the various methodologies, and the best approach to solving a problem using software will often depend on the type of problem. If the problem is well understood and a solution can be effectively planned out ahead of time, the traditional approach may work the best. If, on the other hand, the problem is unique (at

ISSN: 2223-9944, e ISSN: 2223-9553

least to the development team) and the structure of the software solution cannot be easily envisioned, then a more "extreme" incremental approach may work best.

A particular development team may also agree to programming environment details, such as which integrated development environment is used, and one or more dominant programming paradigms, programming style rules, or choice of specific software libraries or software frameworks. These details are generally not dictated by the choice of model or general methodology. The development team is left to choose the best methodology that is the best for the system programming project they are developing. This is well supported by the Microsoft solution framework (MSF) which does not force the developer to use a specific methodology (Waterfall, Agile, etc) but lets them decide what methodology to use. This article is presenting Microsoft solution Frame (MSF) as the best Software Engineering Methodology for Developing System Software and should be adopted.

CONCLUSION

From the sections presentation of this paper, it is clear that the Microsoft solution framework is the best methodology for system software development because from the last section, it showcases that the MSF's philosophy holds that there is no single structure or process that optimally applies to the requirements and environments for all sorts of projects. Therefore MSF supports multiple process approaches, so it can be adapted to support any project, regardless of size or complexity. This flexibility means that it can support a wide degree of variation in the implementation of software engineering processes while retaining a set of core principles and mindsets. The Microsoft Solutions Framework Process Model consists of series of short development cycles and iterations. This model embraces rapid iterative development with continuous learning and refinement, due to progressive understanding of the business and project of the stakeholders. Identifying requirements, product development, and testing occur in overlapping iterations resulting in incremental completion to ensure a flow of value of the project. Each iteration has a different focus and result in a stable portion of the overall system. MSF also has suggestion on how to combine responsibilities such as assigning the program developer to any other role.

ISSN: 2223-9944, e ISSN: 2223-9553

REFERENCES

[1]. Alan, M. D. (2004).*Great Software Debates* pp:125-128 Wiley-IEEE Computer Society Press

[2]. Arcisphere technologies (2012). *"Tutorial: The Software Development Life Cycle (SDLC)"*. Retrieved 2014-08-13.

[3]. Barry, B. (1996*). "A Spiral Model of Software Development and Enhancement*". In: *ACM SIGSOFT Software Engineering Notes* (ACM) 11(4):14-24, August 1986

[4]. Bell, T. E., & Thayer (1996). *Software requirements: Are they really a problem?* Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press.

[5]. Edward, J. B.(2003). *Concepts for Automating Systems Integration* NIST 2003.

[6]. Hughey, D. (2009). *"Comparing Traditional Systems Analysis and Design with Agile Methodologies"*. University of Missouri - St. Louis. Retrieved 11 August 2014.

[7]. Kuhn, D.L. (1989). "*Selecting and effectivelyusing a computer aided software engineering tool"*. Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project.

[8]. McConnell, S. (2014)"7: Lifecycle Planning". *Rapid Development*. Redmond, shington: Microsoft Press. p. 140.

[9]. McConnell, S. (2006). *Software Estimation: Demystifying the Black Art*. Microsoft Press. ISBN 0-7356-0535-1.

[10]. McConnell, S. (2004). *Code Complete, 2nd edition*. Microsoft Press. ISBN 1- 55615-484-4.

[11]. Royce, W. (2012). "Managing the Development of Large Software Systems". Retrieved 11 August 2014.

[12]. Whitten, J. L., Lonnie, D. Bentley, Kevin, C., & Dittman. (2003). *Systems Analysis and Design Methods*. 6th edition. ISBN 0-256-19906-X.

[13]. Whitten (2003). *Rapid Development: Taming Wild Software Schedules*. Microsoft Press. ISBN 1-55615-900-5.

ISSN: 2223-9944, e ISSN: 2223-9553